SCALABLE APPROACHES TO TIME-DEPENDENT PATH AND ROUTING PROBLEMS

A Dissertation
Presented to
The Academic Faculty

By

Edward Yuhang He

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
College of Engineering
H. Milton Stewart School of Industrial and Systems Engineering

Georgia Institute of Technology

August 2021

SCALABLE APPROACHES TO TIME-DEPENDENT PATH AND ROUTING PROBLEMS

Thesis committee:

Prof. Natashia Boland
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Prof. Martin Savelsbergh
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Prof. Michael Hewitt
Quinlan School of Business
Loyola University Chicago

Prof. Alejandro Toriello
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Prof. George Nemhauser
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Date approved: June 24th, 2021

ACKNOWLEDGMENTS

I would like to thank my advisers for their help in preparation of this work – Natashia Boland, who introduced me to this topic and always kept me on my toes with regards to clarity of exposition, George Nemhauser, who advised me during the dark days of the pandemic and gave me the final push I needed to complete this work, and Martin Savelsbergh, who was pivotal in guiding the direction of the research. I would also like to thank my committee members, Michael Hewitt and Alejandro Toriello providing valuable feedback towards completing this work.

Special thanks are due to the friends and colleagues who made this work possible. Luke Marshall for his initial work on Dynamic Discretization Discovery which lay the foundation for much of the dissertation. William Kong and Reem Khir for being great classmates and keeping me up-to-date with everything going on. Kelley Ross and Ritesh Ojha for their help on utilizing the department's computational resources for my experiments. Amanda Ford for being there to deal with all the administrative processes that I could never wrap my head around.

I gratefully acknowledge the support for this work offered by the National Science Foundation under grant award number 1662848, and by the Thomas Johnson Fellowship. Any views and conclusions contained herein are those of the author, and do not necessarily represent the official positions, express or implied, of the funders.

Finally, special appreciation goes to my family who supported me to start this journey. They have truly given me the best upbringing a child could hope for, with their endless love and support. I dedicate this thesis to my girlfriend, Siqi Zhuang, who has supported me through this journey, whose sacrifice and understanding have made all this possible.

TABLE OF CONTENTS

Acknowledgments iii

List of Tables viii

List of Figures x

List of Acronyms xii

Summary xiv

Chapter 1: Introduction and Background 1

1.1 Introduction 1

1.2 Background 2

1.2.1 Time-indexed formulations 2

1.2.2 Solution methods for combinatorial optimization 6

1.2.3 Iterative Refinement Algorithms 11

1.2.4 Dynamic Discretization Discovery 13

1.3 Contributions 14

1.3.1 Overview 14

1.3.2 Time-dependent Shortest Path Problems 15

1.3.3 Time-dependent Shortest Path Problems with Waiting 16

1.3.4 The Service Network Design Problem with Hub Loading and Unloading Capacity 17

Chapter 2: Time-Dependent Shortest Path Problems 19

2.1 Introduction 19

2.2 Problem Description 21

2.3 A Dynamic Discretization Discovery Algorithm for the MDP 26

2.4 A Dynamic Discretization Discovery Algorithm for the MTTP 35

2.4.1 Foundational Theory for the MTTP 37

2.4.2 A DDD Algorithm for the MTTP 43

2.5 Computational Study 51

2.5.1 Instance generation 54

2.5.2 Algorithm configuration 59

2.5.3 Benefits of dynamic discretization discovery 63

2.6 Real-world case study 68

2.7 Concluding Remarks 70

Chapter 3: Time-dependent Shortest Path Problems with Penalties and Limits on Waiting 73

3.1 Introduction 73

3.2 Problem Description and Preliminaries 75

3.2.1 Problem Setting Description and Illustration 75

3.2.2 New Problem Variants 77

3.2.3 Assumptions 80

3.3 Further Cases Equivalent to MATP, MDP or MTTP 81

3.4 Cases that are NP-Hard . . . 88

3.5 Cases of TDSPP-PW Solvable in Polynomial Time via a Time-Expanded Network . . . 94

3.5.1 The Time-Expanded Network . . . 94

3.5.2 Preliminaries . . . 96

3.5.3 Complexity Results . . . 97

3.6 Cases of TDSPP-LW Solvable in Polynomial Time . . . 105

3.6.1 Waiting not allowed ($W = 0$) at a subset of nodes $M \subset N$ . . . 105

3.6.2 Waiting constrained at $M \subset N$ with $M = N \setminus \{n\}$ and a positive limit on total waiting time ($W > 0$) . . . 106

3.7 Summary . . . 114

3.8 Final Remarks . . . 115

Chapter 4: The Service Network Design Problem with Hub Capacity . . . 117

4.1 Introduction . . . 117

4.2 Literature Review . . . 118

4.3 Problem Formulation . . . 121

4.4 Complexity of the SNDP-HC . . . 123

4.5 Integer Program for the SNDP . . . 125

4.6 Integer Program for the SNDP-HC . . . 126

4.7 Methodology . . . 128

4.7.1 Overview . . . 128

4.7.2 Interval Thinking . . . 129

4.7.3 Initialization . . . 131

4.7.4 Creating a Lower-Bound Integer Program 131
4.7.5 Determining Feasibility 136
4.7.6 Adding new time points 144
4.7.7 Complete algorithm and proof 145
4.7.8 Worst-case analysis of Algorithm 4 147
4.8 Handling Infeasibility 149
4.9 Computational Results 150
4.9.1 Instances 150
4.9.2 Algorithm design choices 152
4.9.3 Progression of lower-bound in algorithm 155
4.9.4 Comparison with full integer program 155
4.9.5 Solve times of larger instances 157
4.10 Conclusion 158

Chapter 5: Conclusion 160
5.1 Summary and Novelty of Dissertation 160
5.2 Future Work 161

Appendices 163
Appendix A: Arc travel time functions for the example shown in Figure 2.4 . . . 164

References 166

LIST OF TABLES

2.1 MDP results 64

2.2 MTTP results. 65

2.3 MDP results for large n. 67

3.2 Complexity results for variants of TDSPP-PW obtained so far. Recall that the case of the TDSPP-PW with $\alpha = 0$ is precisely the MTTP, irrespective of the choice of M. 87

3.3 Complexity results for variants of TDSPP-LW obtained so far. 87

3.4 Complexity results for variants of TDSPP-PW. 114

3.5 Complexity results for variants of TDSPP-LW. 114

4.1 Instance features (demand) 151

4.2 Instance features (network) 151

4.3 Progression of lower-bound (Beijing) 157

4.4 Progression of lower-bound (Guangzhou) 157

4.5 Aggregate Results - 10 node (Beijing) 157

4.6 Aggregate Results - 10 node (Guangzhou) 158

4.7 Results - 12 node 158

4.8 Aggregate Results - 15 node (Beijing) 158

4.9 Aggregate Results - 15 node (Guangzhou) 159

4.10 Results - 15 node 159

A.1 Reverse Arc Travel Times at Each BP 165

LIST OF FIGURES

2.1 Example to illustrate the differences between the three objectives. 23

2.2 Example Network 1 27

2.3 Arc Travel Times at Each BP 27

2.4 Arc Travel Times 27

2.5 Procedure to generate the ABSPT corresponding to $(1, 0)$. 29

2.6 TEN in each iteration for the example in Figure 2.2 (note that an ABSPT may satisfy multiple criteria in the legend). 36

2.7 Illustration of a timed path with 3 periods of waiting in thick gray and 3 travel subpaths in thick black, in the TEN for a network consisting only of the arcs $(1, 2), (2, 3), (3, 4)$ and $(4, 5)$. 38

2.8 Forwards waiting arcs: Cases 1 and 2(a). 47

2.9 Backwards waiting arc: Case 2(b). 47

2.10 Lower bound paths using waiting arcs allow possible optimal paths to be represented. 48

2.11 Examples of network types. 56

2.12 Plots of different travel time function types 58

2.13 Comparison of breakpoint selection schemes on 120 instances of the MDP with $n = 50$, $T = 40, 60$ 61

2.14 Comparison of breakpoint selection schemes on 120 instances of the MTTP with $n = 50$, $T = 40, 60$ 62

2.15 Map and corresponding network representation of the section of Midtown and North Atlanta. 69

2.16 Travel times for four different arcs in Atlanta. 70

2.17 An example where the minimum duration path differs from the earliest arrival/latest departure path. 71

3.2 The optimal solution to the TDSPP-PW with $\alpha > 1$: the case of $M = N \setminus \{n\} = \{1, 2, 3\}$ is shown as solid lines and the case of $M = N \setminus \{1\} = \{2, 3, 4\}$ is shown as dashed lines. 79

3.3 Example Network 2. 90

3.4 Travel time functions for Example Network 2. 90

4.1 Effect of initial discretization (Beijing) 153

4.2 Effect of initial discretization (Guangzhou) 153

4.3 Effect of various design choices (Beijing) 156

4.4 Effect of various design choices (Guangzhou) 156

LIST OF ACRONYMS

ABSPT *Arc-completed Backwards Shortest Path Tree*

AFSPT *Arc-completed Forwards Shortest Path Tree*

BP *Breakpoint*

BSPT *Backwards Shortest Path Tree*

DDD *Dynamic Discretization Discovery*

EPLP *Exact Path Length Problem*

FIFO *First-In First-Out*

FSPT *Forwards Shortest Path Tree*

MATP *Minimum Arrival Time Problem*

MCNF *Multi-Commodity Network Flow*

MDP *Minimum Duration Problem*

MILP *Mixed-Integer Linear Programming*

MTTP *Minimum Travel Time Problem*

SNDP *Service Network Design Problem*

SNDP-HC *Service Network Design Problem with Hub (Loading and Unloading) Capacity*

SP *Shortest Path*

TDSP *Time-Dependent Shortest Path*

TDSPP *Time-Dependent Shortest Path Problem*

TDSPP-LW *Time-Dependent Shortest Path Problem with Limited Waiting*

TDSPP-PW *Time-Dependent Shortest Path Problem with Penalized Waiting*

TEN *Time-Expanded Network*

TENL *Time-Expanded Network with Associated Arc Lengths*

TSP *Traveling Salesman Problem*

UTT *Underestimated Travel Time*

SUMMARY

This dissertation is based in part on the previously published articles in He et al. (2019) and He et al. (2020).

Many combinatorial optimization problems involve an aspect of time, whether due to changing conditions, updated information, or being under a continuous decision-making process. A popular domain for time-based decision making is logistics, where it is not only important to decide along which routes products need to be transported, but also when the transportation should occur. The timing for these problems is critical to take advantage of changing operational conditions such as traffic, capacity, and costs. Unfortunately, introducing time as an additional dimension significantly increases the size of models over their static counterparts. In fact, solving such models is often done via heuristics as existing solution methods are computationally intractable. This dissertation aims to provide insights into how scalable solution methodologies can be developed for such problems in path and routing problems.

Chapter 1 introduces time-indexed formulations, which are a result of time-based decision making, and time-expanded networks which is both a visualization tool and a related abstract concept. Following that, several solution methodologies for combinatorial optimization problems are described, along with why they are not always sufficient for all such problems. The history of iterative refinement algorithms, and in particular *Dynamic Discretization Discovery* (DDD), which is the core method in this dissertation, is presented. Finally, a highlight of the goals and contributions of this dissertation is provided.

Chapter 2 shows how DDD can be applied to the minimum duration time-dependent shortest path problem to significantly improve the run-time over existing methods. It also introduces the first polynomial-time algorithm for a related problem, the minimum travel time time-dependent shortest path problem, as well as how DDD can also be used for this problem to reduce the search space.

Chapter 3 builds upon the theory of time-expanded networks from Chapter 2 to provide new computational complexity results and polynomial-time algorithms on a large class of time-dependent shortest path problems with waiting. While this chapter does not directly use DDD, the new algorithms solve time-dependent shortest path problems identical to those in Chapter 2 as a subroutine and thus can take advantage of DDD.

Chapter 4 explores how DDD can be applied to the *Service Network Design Problem with Hub (Loading and Unloading) Capacity* (SNDP-HC), an NP-hard problem where even determining feasibility is an open question. Instances of SNDP-HC motivated by real-life problems result in integer programming formulations consisting of billions of constraints and hundreds of millions of variables. Using DDD, the size of the integer programming formulations can be reduced significantly, which enables small and medium-sized instances of SNDP-HC to be solved exactly in a reasonable time. For larger instances, this approach serves as a good starting point for the development of future algorithms.

Finally, Chapter 5 summarizes the findings in this dissertation and explores avenues for future research of DDD.

CHAPTER 1
INTRODUCTION AND BACKGROUND

1.1 Introduction

Solving large scale path and routing problems with a temporal aspect is often done with the aid of a time-expanded network which allows decisions to be made about not only *what* should happen but also *when* it should happen. Complete time-expanded networks that account for all possibilities are often too large to be solved with current computing power. Instead, it is standard to apply a time discretization in which events can only occur at certain points in time such as every hour, day, or week. When the discretization (set of time points for which decisions can be made) is too coarse, the solution found by solving such a problem may be suboptimal compared to solutions found by solving a problem with a finer discretization. Even worse, it is possible for the problem to be infeasible, even when feasible solutions exist when considering a finer discretization. However, using a finer discretization results in problems that are significantly harder to solve. In this dissertation, we explore a recent concept known as *Dynamic Discretization Discovery* (DDD), which aims to combine the best of both worlds by creating time-expanded networks that have the size of a coarse discretization but the accuracy of a fine discretization. This is done using an iterative process that discovers time points that are relevant to finding an optimal solution and includes them in the time-expanded network. Applications to the service network design problem in Boland et al. (2017) has shown this method to be very promising, achieving performance improvements over other existing methods of up to 28% on real-life instances. In particular, we apply the concept to three problems, the *Minimum Duration Problem* (MDP), the *Minimum Travel Time Problem* (MTTP) and the *Service Network Design Problem with Hub (Loading and Unloading) Capacity* (SNDP-HC).

1.2 Background

1.2.1 Time-indexed formulations

This subsection provides background on time-indexed formulations, which is the formulation of choice for the problems presented in this dissertation. To start, some classical problems which naturally have time-indexed formulations are presented along with possible alternative formulations, to highlight when a time-indexed formulation might be preferred.

Definition 1. *A time-indexed formulation is a mathematical programming formulation in which there exists at least one variable that is indexed by time. In other words, subject to renaming, there exists variables x_{it} where $i \in I$, $t \in T$ and t corresponds precisely to a time at which the decision x_i can occur.*

One of the earliest examples of time-indexed formulations was given for the schedule-sequencing problem in Bowman (1959). A closely related, and more classic problem is the single-machine scheduling problem (Gupta and Kyparisis 1987, Belouadah et al. 1992), where time-indexed formulations are often used (Sousa and Wolsey 1992).

Definition 2. *Given is a set of jobs I to be scheduled on a single machine that can only run one task concurrently. Each job $i \in I$ has a release time r_i, processing time p_i, and deadline d_i. The* single-machine scheduling problem *aims to find a schedule of jobs such that each job starts after its release time, processes for a consecutive p_i units of time by itself. The objective is typically related to the completion time of each task C_i, for example, it could be to minimize the sum of tardiness $\sum_{i \in I} T_i$, where $T_i = \max(0, C_i - d_i)$.*

For ease of exposition, let us consider the simplified scenario in which the jobs have identical processing times, i.e., $p_i = p$, for all $i \in I$. For the minimum tardiness objective, there is no explicit time-indexing in the objective since variables C_i are only indexed by the job i. However, one natural way of defining C_i, while also being able to specify the required constraints, would be to use binary variables x_{it} which indicate whether job i begins at time

t. Without loss of generality, assume all jobs occur within a time horizon $[0, T]$, where T is any upper bound on the latest completion time. Using this, a possible mixed-integer linear programming formulation can be given as:

$$\min_{x,T,C} \sum_{i \in I} T_i \tag{1.1}$$

$$\text{subject to:} \quad T_i \geq C_i - d_i \qquad \forall i \in I, \tag{1.2}$$

$$\sum_{t=\tau}^{\tau+p-1} \sum_{i \in I} x_{it} \leq 1 \qquad \forall \tau \in [0, T], \tag{1.3}$$

$$p + \sum_{t=r_i}^{T} t x_{it} = C_i \qquad \forall i \in I, \tag{1.4}$$

$$\sum_{t=r_i}^{T} x_{it} = 1 \qquad \forall i \in I, \tag{1.5}$$

$$x_{it} \in \{0, 1\} \qquad \forall i \in I, t \in [r_i, T], \tag{1.6}$$

$$T_i \geq 0 \qquad \forall i \in I. \tag{1.7}$$

The objective function in (1.1) minimizes the total tardiness, where tardiness is defined by (1.2) and (1.7). (1.3) enforces that at most one job is processing concurrently. (1.4) defines the completion time of job i. (1.5) and (1.6) ensures that each job is completed. This formulation is very intuitive and the number of variables is $\mathcal{O}(|I|T)$ and the number of constraints is $\mathcal{O}(|I| + T)$.

It is also possible, though slightly more difficult, to formulate this problem without time-indexing. The parameters of the problem as well as the objective both directly relate to time, however, time can also be captured implicitly. For example, let S be the set of all possible sequences of jobs. Then for each $s \in S$, the optimal timing of jobs can be calculated by scheduling jobs as soon as possible, at the earliest of the previous job finishing time or the release time of the current job. Thus, once the tardiness for each s is calculated,

the timings are implicitly captured in s and do not need to be modeled explicitly. Of course, such a formulation is exponential in nature due to the number of possible sequences. However, that does not mean that this formulation is not useful since there may be conditions such as precedence relationships between jobs, or solution methods such as column generation that lend themselves to such a formulation. The next problem has these properties, leading to a situation where this type of formulation is preferred over time-indexed formulations.

One classic problem in operations research is the *Traveling Salesman Problem* (TSP), for which a simplified definition is provided below.

Definition 3. *Given a complete network $D = (N, A)$ consisting of nodes $N = \{1, 2, \ldots, n\}$ and arcs A, with travel time c_{ij} for traversing arc $(i, j) \in A$. Find a shortest cycle that visits every node exactly once.*

The standard integer programming formulation for the TSP provided in Dantzig et al. (1954) uses binary variables x_{ij} that indicates whether the arc (i, j) is used in the selected cycle.

$$\min_x \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{1.8}$$

subject to:

$$\sum_{i\in\delta^-(j)} x_{ij} = 1 \qquad \forall j \in N, \tag{1.9}$$

$$\sum_{j\in\delta^+(i)} x_{ij} = 1 \qquad \forall i \in N, \tag{1.10}$$

$$\sum_{i\in S}\sum_{j\in S, j\neq i} x_{ij} \leq |S| - 1 \qquad \forall S \subset \{1, \ldots, n\}, |S| \geq 2 \tag{1.11}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i, j) \in A \tag{1.12}$$

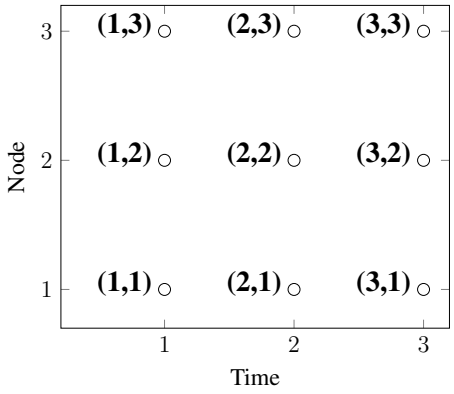
(1.8) minimizes the sum of travel times, subject to visiting each node exactly once ((1.9), (1.10) and (1.12)). (1.11) prevents subtours in which the solution consists of more

than one disjoint cycles from occurring. While there are exponentially many constraints in (1.11), row generation approaches can be used to greatly reduce the number of constraints used in practice.

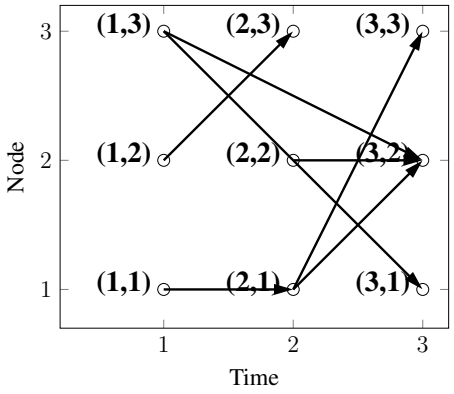
Note that the traveling salesman problem becomes a variant of the single-machine scheduling problem if we consider visiting a node as a 'job' and processing times c_{ij} now dependent on both the current job j and the previous job i. Release times are identically zero and deadlines are not considered. Instead, the objective is to minimize the latest completion time. Using this interpretation, a time-indexed formulation for the TSP is also possible by letting variables x_{ijt} indicate ending job i and starting job j at order t in the cycle. For the full formulation (of the more general time-dependent traveling salesman problem) with $\mathcal{O}(n)$ constraints, see Fox et al. (1980).

In both machine scheduling problems and TSP, using the time-indexed formulation leads to smaller integer programming formulations. Other formulations may have the advantages of exploiting problem structure, being amenable to row or column generation, or having tighter relaxations. It appears to be a general trend that time-indexed formulations can be significantly smaller but often have poor linear programming relaxations.

Another idea that is central to time-indexed formulations is the idea of a *Time-Expanded Network* (TEN), which are networks with an additional temporal dimension. For time-indexed variables x_{it}, the space for t can be viewed as points along the horizontal axis, while the space for i can be viewed as points along the vertical axis. For example, let $i \in \{1, 2, 3\}$ represent possibly nodes or decisions and $t \in \{1, 2, 3\}$ represent times at which decisions are made, then the corresponding TEN is shown in Figure 1.1a. Suppose that there are also transitions, that is a decision made at a certain time affects decisions made in the future such as the cost of decisions or restrictions to which decisions could be made, these are shown as arcs in the TEN (see Figure 1.1b). For the rest of this dissertation, the space of decisions I is tied to the spatial dimension, which will be a set of nodes in the underlying network of the problem being considered.



(a) Example Time-expanded Network



(b) Example Time-expanded Network with Transitions

1.2.2 Solution methods for combinatorial optimization

While there have been tremendous advances in computational power, there has also been a comparable increase in the complexity of problems faced. Researchers and practitioners constantly try to push the boundaries of what is possible by including greater detail or considering greater scope, both of which lead to harder problems. When given a combinatorial optimization problem, several solution methods are available. When the combinatorial optimization is formulated as a mixed-integer linear programming problem, one of the basic methods is to use a vanilla branch-and-bound algorithm using a linear programming relaxation, which works very well for small problem sizes. Enhancements such as adjusting the relaxation (such as using a Lagrangian relaxation instead of using a linear programming relaxation), adding cutting planes (i.e. branch-and-cut), and changing branching rules can be used to speed up the algorithm. Alternatively, the problem size can be reduced by row generation or column generation on the linear programming relaxation. The success of which largely depends on the problem properties. When all else fails, heuristics and metaheuristics can be used to find suboptimal but useful solutions. In this subsection, we briefly review solution methodologies for mixed-integer linear programming. Of particular interest are row generation, column generation, and aggregation techniques, as they have similarities to *Dynamic Discretization Discovery* (DDD), which is the solution method of

interest in this dissertation.

General framework for solving mixed-integer linear programming problems

Although there are numerous ways of solving mixed-integer linear programming problems, many methods use *relaxations*.

Definition 4. *A* relaxation *for the optimization problem* $\min_{x \in X} f(x)$ *is any optimization problem* $\min_{y \in Y} g(y)$ *such that* $X \subset Y$ *and* $g(x) \leq f(x)$ *for* $x \in X$.

A common framework is:

1. Start with a *relaxation* of the original problem or one of its dual. This relaxation should be easier than the problem that needs to be solved.

2. By solving the relaxation, a (primal or dual) bound on the optimal objective value of the original problem is obtained.

3. Using information from step 2 and/or by performing additional computation, *identify a tighter relaxation*. This is usually done by solving a *separation problem* or *pricing problem*.

4. Repeats steps 2 and 3 with the new relaxation until either the relaxation is infeasible, in which case no feasible solution exists for the original problem, or the optimal solution of the relaxation can be transformed to a feasible solution for the original problem and has the same objective value in the original problem and relaxation. The feasible solution found is then optimal for the original problem.

The selection of the relaxation to use and how tighter relaxations are identified are the distinguishing factor between solution methodologies.

Branch-and-bound

The *branch-and-bound algorithm* (first presented in Land (1960)) is the cornerstone of solving integer and mixed-integer linear programming problems. Due to the combinatorial nature of such problems, it is possible that exponentially many solutions need to be considered to find the optimal solution. However, the clever use of lower and upper bounding greatly reduces the number of solutions considered in practice.

Definition 5. *Given a minimization mixed-integer linear program, the* (basic) branch-and-bound algorithm *is as follows:*

1. *Start with an unresolved node containing the mixed-integer linear program.*
2. *While there are unresolved nodes, do:*
   (a) *Solve the linear programming relaxation of the problem at the node. Set the current node to resolved.*
   (b) *If the objective value is higher than the incumbent value, there is no more exploring to be done at this node, continue searching other unresolved nodes.*
   (c) *If the solution has non-integer values for integer variables, branch into two unresolved nodes by dividing the search space into two mutually exclusive, collectively exhaustive (with respect to integer solutions) regions by adding additional constraints (cuts) to the mixed-integer linear program at the current node.*
   (d) *Otherwise set the current solution as the incumbent.*
3. *Return the incumbent solution if it exists, otherwise, there are no feasible solutions.*

For the basic branch-and-bound algorithm, the relaxation used is the linear programming relaxation and tighter relaxations are generated by identifying a fractional variable $a < x < a + 1$ (where $a \in \mathbb{Z}$) that should instead be integer-valued and excluding the variable from taking values in $(a, a + 1)$ in subsequent relaxations.

Row Generation

Row generation, also known as cut generation, starts by discarding a subset of the constraints to form the initial relaxation. It exploits the fact that when the constraint matrix is column rank-deficient, only a few constraints are needed to define the optimal solution. Additional constraints are found iteratively and tighten the relaxation until the optimal solution is found. One specialized example of this is Benders decomposition (Benders 1962).

Definition 6. *Given a minimization linear program, a* row generation *approach does the following:*

1. *Start with a linear program and discard a subset of the constraints to form an initial relaxation.*
2. *While the feasibility and optimality status of the problem is unknown, do:*
   (a) *Solve the relaxation. If it is infeasible, the original problem was infeasible.*
   (b) *Solve a sub-problem that either finds a (removed) constraint that is violated by the solution of the relaxation or a constraint that uses additional variables not determined by the relaxation. This constraint is then added to the relaxation to tighten it. If no such constraint is found, then we have found an optimal solution to the original problem.*
3. *Return the solution, if it exists, otherwise there are no feasible solutions.*

Row generation is typically used either when the problem has a decomposition structure (such as that of Benders decomposition), or when the constraint matrix is highly column rank-deficient (in the case of the traveling salesman problem (Padberg and Rinaldi 1987)). There is also another use case where there exists a subset of constraints that significantly increase the difficulty of the problem, but if only a couple of constraints from the subset are present the problem becomes manageable e.g. by incorporating Lagrangian relaxation

techniques (Barcelo et al. 1990). To see how this can be applied to mixed-integer linear programming problems, see McDaniel and Devine (1977), Cote and Laughton (1984), Chu and Xia (2004).

Column generation

Column generation starts by discarding a subset of the columns (can also be viewed as variables) to form the initial formulation. It can be viewed as applying row generation to a dual problem. It exploits the fact that when the constraint matrix is row rank-deficient, only a few variables need to be explicitly determined at an optimal solution. Additional columns are found iteratively and tighten the upper-bound formulation until the optimal solution is found. One specialized example of this is Dantzig-Wolfe decomposition (Dantzig and Wolfe 1960).

Definition 7. *Given a minimization linear program, a* column generation *approach does the following:*

1. *Start with a linear program and discard a subset of the variables to form an initial upper-bound formulation. This is usually done in a manner such that a feasible solution is available for the upper-bound formulation.*
2. *While the optimality status of the problem is unknown, do:*
   (a) *Solve the upper-bound formulation.*
   (b) *Solve a pricing sub-problem that finds a (removed) column with a negative reduced cost. This column is then added to the upper-bound formulation to improve it. If no such constraint is found, then we have found an optimal solution for the original problem.*
3. *Return the optimal solution.*

Column generation is typically used either when the problem has a decomposition structure (such as that of Dantzig-Wolfe decomposition), or when the constraint matrix is highly row rank-deficient (such as the cutting-stock problem (Vance et al. 1994) or vehicle routing problem (Desrochers et al. 1992)). To see how this can be applied to mixed-integer linear programming problems, see Wilhelm (2001), Nemhauser (2012).

Heuristics

While this dissertation focuses on exact methods, it is important to mention that there are plenty of inexact methods that work well in practice: local search heuristics (Groër et al. 2010), aggregation/disaggregation methods (Rogers et al. 1991, Espinoza et al. 2008), genetic algorithms (Sivanandam and Deepa 2008), tabu search (Glover and Laguna 1998) and approximation algorithms (Williamson and Shmoys 2011).

1.2.3 Iterative Refinement Algorithms

While existing approaches such as column generation and row generation work well for a wide variety of problems, they are not capable of shrinking both the column and row dimensions of the constraint matrix simultaneously. Moreover, if the rank of the constraint matrix is high, many iterations would be required before convergence. For certain time-indexed formulations, more powerful decomposition methods are needed.

Iterative refinement is a specialized framework for solving time-indexed formulations which aim to reduce both the column and row dimensions of the constraint matrix simultaneously. Iterative Refinement works by aggregating nodes on the time-expanded network for which the time-indexed formulation is based. The specific aggregation is across time, where different nodes that have the same spatial index, but different temporal indices are treated as one node which captures properties from the aggregate nodes. The partial time-expanded network is then used as a basis for a smaller time-indexed formulation that yields a bound on the original objective function value. How this is done depends on the spe-

cific underlying problem. Methods that terminate with such a solution, without pursuing optimality, are simply called time-window discretization methods (Hane et al. 1995, Jarrah et al. 2000). Iterative refinement methods further improve upon this lower-bound, identifying areas of the partial time-expanded network for disaggregation by using information from the lower-bound solution and solving additional optimization problems. The philosophy is that large regions of the time-expanded network are not required to determine the optimal solution and will thus not need to be explored, leading to performance gains.

One of the key differences between iterative refinement algorithms and the previous algorithms mentioned is that the auxiliary optimization problems do not share the same variable space as the original optimization problem or any of its dual. Instead, because of aggregation, the auxiliary optimization problems can be viewed as a projection of the space of variables to a lower-dimensional space. This yields the advantage of working with inherently lower-dimensional problems.

Literature Review of Iterative Refinement

While there has been limited literature on iterative refinement schemes, the usage has been relatively successful. A brief overview of the history of iterative refinement is provided below.

Wang and Regan (2002) used an iterative refinement scheme to iteratively partition time-windows for the local truckload pickup-and-delivery problem. While they also utilized a similar lower-bound construction that is updated iteratively, they do not provide any *convergence guarantee*. They do note that the lower-bounds produced are tight and may even coincide with the optimal solution.

Wang and Regan (2009) provides the first guarantee of convergence for iterative refinement algorithms on time-expanded networks when applied to the traveling salesman problem with time-windows. They show that as the number of iterations tends to infinity, the algorithm converges towards the optimal solution. They do not show that the algorithm

has *finite convergence*.

Dash et al. (2012) builds upon Wang and Regan (2009) by using the iterative refinement algorithm to *initialize a branch-and-cut algorithm*. While this results in finite convergence, the approach is significantly different from using iterative refinement until optimality.

Finally, Boland et al. (2017) provides the first version of iterative refinement that has *finite convergence guarantee* and is capable of being used as a stand-alone tool to find the optimal solution, the authors coined the method that has these properties as *Dynamic Discretization Discovery* (DDD). More details on their implementation are provided in Chapter 4. Other applications of DDD include Lagos et al., Hewitt (2019), Scherr et al. (2020), Vu et al. (2020), Marshall et al. (2021). This recent literature has shown that the application of DDD results in significant improvements over existing benchmarks and robustness over the choices of parameters of the problem.

This dissertation extends the ideas of DDD to various path and routing problems to yield scalable approaches.

1.2.4 Dynamic Discretization Discovery

The distinguishing features of *Dynamic Discretization Discovery* (DDD) come from several invariant properties that are carefully maintained throughout the iterations of the algorithm. First, the finite convergence guarantee for DDD relies on Property 1 to hold.

Property 1. *The sequence of partial time-expanded networks converges to the original time-expanded network. This is done by ensuring that once node aggregations are selected in the first iteration, subsequent iterations only disaggregate nodes. Disaggregations are done in a careful manner such that if all nodes are disaggregated, the resulting time-expanded network, including arcs, should be the same as the original. This results in identical optimization problems for both the original and disaggregated time-expanded network, thus returning the original objective function value and solution.*

Clearly, Property 1 alone does not result in a useful algorithm since it only says that in

the worst case, the algorithm will resort to solving the original problem directly. Additional properties such as monotonic improvement are required to motivate early termination of the algorithm.

Property 2. *The optimization problems solved provide non-decreasing lower-bounds on the original objective function value.*

If at any stage, a feasible solution for the original problem that matches the current lower-bound can be found, then it must be optimal for the original problem. Property 2 ensures that progress towards feasibility is being made while maintaining optimality. This is important as the hope is that feasibility occurs relatively early in the algorithm, where the cost of solving the optimization problems is low to achieve performance gains.

1.3 Contributions

In this section, an overview of the overarching ideas for the dissertation is presented, followed by separate subsections that detail the contributions of each chapter of the dissertation.

1.3.1 Overview

This dissertation aims is to investigate the *Dynamic Discretization Discovery* (DDD) framework to develop efficient and scalable iterative algorithms for time-dependent path and routing problems. To begin, I investigate different objective functions for the *Time-Dependent Shortest Path Problem* (TDSPP), which is a variant of the traditional shortest path problem where the time to traverse an arc depends on when the traversal occurs. Much of the literature is concerned with the *Minimum Arrival Time Problem* (MATP), where the objective is to arrive as early as possible. I present a DDD algorithm for the *Minimum Duration Problem* (MDP), where the objective is to minimize the difference between departure at the origin and arrival at the destination. This problem was just recently known to

be polynomial-time solvable (Foschini et al. 2014) and, due to the structure of the solution algorithm, one of the simplest problems where DDD can lead to significant improvements. A related problem is the *Minimum Travel Time Problem* (MTTP), where the objective is to minimize the total time spent along arcs. Insights from exploring the MDP are used to provide the first polynomial-time algorithm for the *Minimum Travel Time Problem* (MTTP) as well as create a corresponding DDD algorithm. By looking at time-dependent shortest paths with varying objectives, it was clear that there was an opportunity to consolidate these problems into more general classes of time-dependent shortest path problems with waiting. This was done in two different ways, either by introducing waiting in the objective function or by including waiting as a constraint. Doing so revealed several unexplored classes of time-dependent shortest path problems for which new complexity results, as well as new solution algorithms, are presented. Finally, we return to the *Service Network Design Problem* (SNDP) where DDD was first applied and incorporate hub loading and unloading constraints, which increases the modeling capabilities for last-mile delivery in a city logistics setting.

1.3.2 Time-dependent Shortest Path Problems

Finding a shortest path in a network is a fundamental optimization problem. Chapter 2 focuses on settings in which the travel time on an arc in the network depends on the time at which traversal of the arc begins. In such settings, reaching the destination as early as possible is not the only objective of interest. Minimizing the duration of the path, i.e., the difference between the arrival time at the destination and the departure from the origin, and minimizing the travel time along the path from origin to destination, are also of interest.

Dynamic Discretization Discovery (DDD) algorithms are used to efficiently solve such time-dependent shortest path problems with piecewise linear arc travel time functions. The algorithms operate on partially time-expanded networks in which arc costs represent lower bounds on the arc travel time over the subsequent time interval. A shortest path in this

partially time-expanded network yields a lower bound on the value of an optimal path. Upper bounds are easily obtained as byproducts of the lower bound calculations. The algorithms iteratively refine the discretization by exploiting breakpoints of the arc travel time functions. In addition to time discretization refinement, the algorithms permit time intervals to be eliminated, improving lower and upper bounds, until, in a finite number of iterations, optimality is proved.

The main contributions of this chapter are the first polynomial-time algorithm for the minimum travel time problem as well as carefully designed *Dynamic Discretization Discovery* (DDD) algorithms for both the *Minimum Duration Problem* (MDP) and *Minimum Travel Time Problem* (MTTP). The new algorithms greatly reduce the search space (up to $1000\times$ for the MDP and up to $5\times$ for the MTTP) that is required in the baseline polynomial-time algorithms for these problems. This reduction directly results in reduced computation time of up to $200\times$ for the MDP and $2\times$ for the MTTP. In addition, the performance gains increase with the length of the time horizon and the size of the network, making the algorithms highly efficient and scalable.

1.3.3 Time-dependent Shortest Path Problems with Waiting

Waiting at the right location at the right time can be critically important in certain variants of time-dependent shortest path problems. Chapter 3 begins by first defining two new classes of time-dependent shortest path problems that incorporate waiting, either by considering a penalty on waiting, which we call the *Time-Dependent Shortest Path Problem with Penalized Waiting* (TDSPP-PW), or a limit on the total time spent waiting at a given subset of the nodes, known as the *Time-Dependent Shortest Path Problem with Limited Waiting* (TDSPP-LW). Both classes of problems are capable of generalizing the problems explored in Chapter 2: *Minimum Arrival Time Problem* (MATP), *Minimum Duration Problem* (MDP) and *Minimum Travel Time Problem* (MTTP) via the selection of parameters. The parameters control the magnitude of the penalty on waiting for the TDSPP-PW and the

limit on waiting for the TDSPP-LW, as well as the subset of nodes on which waiting is penalized or is constrained. In addition to generalizing existing problems, other selections of the parameters reveal several interesting cases that have not yet been explored in existing literature. The computational complexity of all the remaining cases is proven and some cases are shown to be NP-hard via polynomial reductions, while others are shown to be solvable in polynomial time, in which case a polynomial-time algorithm is provided. While DDD is not mentioned explicitly, some of the algorithms presented use the algorithms in Chapter 2 as subroutines. In addition, the core ideas of the proofs rely on exploiting knowledge of the underlying time-expanded networks.

1.3.4 The Service Network Design Problem with Hub Loading and Unloading Capacity

The *Service Network Design Problem* (SNDP) is commonly used to solve tactical decision-making problems regarding package routing decisions. When hubs are located in areas with plentiful space, hubs are modeled to have unlimited loading and unloading capacity. However, in a city logistics setting, it is necessary to model loading and unloading explicitly due to the lack of space and short time window of operations. The difficulty of the *Service Network Design Problem with Hub (Loading and Unloading) Capacity* (SNDP-HC) arises from the presence of an additional time-index on the modeling variables, resulting in doubly time-indexed variables and a significantly larger integer programming problem. I present an exact algorithm for the *Service Network Design Problem with Hub (Loading and Unloading) Capacity* (SNDP-HC), which is the first application of DDD to a problem with doubly time-indexed variables and constraints. While SNDP-HC is significantly harder to solve than SNDP, there are also greater benefits in applying DDD. In particular, the presence of two time-indexes results in a reduction by a squared factor in both the number of variables and the number of constraints rather than a linear factor when starting with a coarser discretization. Computational experiments on real-world data show that the

algorithm is capable of solving instances that were unsolvable via existing methods.

CHAPTER 2

TIME-DEPENDENT SHORTEST PATH PROBLEMS

2.1 Introduction

Finding a shortest path between two locations in a network is a critical component of many algorithms for solving transportation problems. There is growing interest in settings in which the travel time along an arc in the network is a function of the time the travel starts. Such *time-dependent travel times* are typically a result of congestion. As is commonly done, we assume that travel times on arcs satisfy the *First-In First-Out* (FIFO) property: it is impossible to arrive at the end of an arc earlier by starting travel along the arc later.

Several different objective functions arise when seeking a path in a network with time-dependent travel times. In the simplest version of the problem, the departure time at the origin is given and a path that reaches the destination as early as possible is sought. We refer to this as the Minimum *Arrival Time* Path Problem (MATP). The first approach developed to solve this problem is based on the Bellman-Ford algorithm (Bellman 1958, Ford and Fulkerson 1962) and is described by Cooke and Halsey (1966). An overview of other methods for this variant is given by Dean (2004b).

In this chapter, we focus on two different variants: (1) the problem of finding a path such that the difference between the departure time at the origin and the arrival time at the destination is as small as possible, and (2) the problem of finding a path such that the total travel time from the origin to the destination is as small as possible. We refer to these variants as the Minimum *Duration* Time-Dependent Path Problem (MDP) and the Minimum *Travel Time* Time-Dependent Path Problem (MTTP), respectively. Both of these problems allow waiting at nodes in the network, which makes them much more complicated than the MATP.

The MDP arises in many contexts. It has been studied, for example, in the context of traffic networks (Demiryurek et al. 2011), and it has even arisen in the analysis of social networks (Gunturi et al. 2012). The MTTP is especially relevant when seeking to reduce emissions in urban environments, since a vehicle's emissions along a given section of road are approximately proportional to its travel time on that section, for the range of speeds typically used in urban settings (see, for example, the emissions formula used by Jabali et al. (2012) in the time-dependent vehicle routing context). A variant of MTTP in which waiting is only allowed at a given subset of nodes has been considered by Li et al. (2017). Their solution approach is based on function compositions and relies on travel time function approximations to counteract the exponential nature of function compositions.

Recently, Boland et al. (2017) introduced a dynamic discretization discovery algorithm for solving the continuous time minimum cost service network design problem, which uses integer programming formulations over partially time-expanded networks. The key to the approach is that it discovers, in an efficient way, times to include in the time-expanded network so that the associated integer program yields a solution that is optimal for the continuous time problem. It does so by solving a sequence of (small) integer programs, each a function of the subset of times used to define the partially time-expanded network. The partially time-expanded networks are carefully designed to result in integer programs that are tractable in practice, and that yield a dual (lower) bound on the optimal continuous-time value. Once the *right* (very small) subset of times is discovered, the resulting integer programming model yields the continuous-time optimal value. The idea of starting with a coarse time discretization to obtain a lower bound and iteratively refining the time discretization has been applied previously in the context of the Traveling Salesman Problem with Time Windows (Wang and Regan 2002, 2009, Dash et al. 2012), but there are significant differences in terms of the time points considered in a refinement step, the selection of the time points to add in a refinement step, and how the time discretizations are used to producing a feasible solution.

In this chapter, we present and analyze dynamic discretization discovery algorithms for the MDP and the MTTP when the travel times on arcs are given by piecewise linear functions. The algorithms solve a sequence of shortest path problems in partially time-expanded networks, each of which provides a successively better lower bound on the value of the continuous-time problem. The time-space nodes in the partially time-expanded networks are derived from breakpoints of the piecewise linear travel time functions. The shortest path at each iteration suggests new times to include in the time-expanded network, and its true objective value (its duration or travel time) can easily be calculated to yield an upper bound on the value of the continuous-time problem. The sequence concludes when the length of the shortest path in the current partially time-expanded network matches the best upper bound.

For the MDP, it was established only recently that an algorithm polynomial in the number of travel time function breakpoints exists (Foschini et al. 2014). We extend this result to the MTTP, and, thus, provide the first polynomial time algorithm for the MTTP. However, our key contribution is the development of algorithms that, through dynamic discretization discovery, investigate only a small fraction of the travel time function breakpoints in the search for an optimal path and a proof of its optimality.

The remainder of the chapter is organized as follows. In Section 2.2, we formally introduce the MDP and MTTP and briefly discuss the relevant literature. In Section 2.3, we describe our algorithm for the MDP and illustrate its operation on a small instance. We then present an algorithm for the MTTP in Section 2.4. In Section 2.5, we present the results of a comprehensive computational study.

2.2 Problem Description

We are given a directed network $D = (N, A)$ with $N = \{1, 2, \ldots, n\}$ and $A \subseteq N \times N$, a time interval $[0, T]$, and piecewise linear travel times $c_{i,j}(t) > 0$ for $t \in [0, T]$ satisfying the *First-In First-Out* (FIFO) property for arcs $(i, j) \in A$. Satisfying the FIFO property, in

the case of piecewise linear travel time functions, is equivalent to having the slopes of the linear pieces be at least -1. Without loss of generality, we let node 1 be the origin and node n be the destination.

A *timed path* $P = ((i_1^P, t_1^P), (i_2^P, t_2^P), \ldots, (i_{m(P)}^P, t_{m(P)}^P))$ consists of a sequence of $m(P)$ node and time pairs, where its node sequence, $(i_1^P, i_2^P, \ldots, i_{m(P)}^P)$, induces a path in D from i_1^P to $i_{m(P)}^P$, and where each node in the sequence, i_k^P, has an associated time, t_k^P, representing the time at which the traversal of arc (i_k^P, i_{k+1}^P) begins, for each $k = 1, \ldots, m(P) - 1$, and the arrival time at node i_k^P, for $k = m(P)$. The sequence of times is thus required to satisfy $t_{k+1}^P \geq t_k^P + c_{i_k^P, i_{k+1}^P}(t_k^P)$ for $k = 1, \ldots, m(P) - 1$. If $t_k^P > t_{k-1}^P + c_{i_{k-1}^P, i_k^P}(t_{k-1}^P)$ for any $k \geq 2$, (or if $t_1^P > 0$), then we say that there is waiting at node i_k^P. If $t_k^P = t_{k-1}^P + c_{i_{k-1}^P, i_k^P}(t_{k-1}^P)$ for all $k \geq 2$ then we say that P is *waiting-free*. If P starts at node $i_1^P = 1$ with $t_1^P \geq 0$ and ends at node $i_{m(P)}^P = n$ with $t_{m(P)}^P \leq T$ then we say that P is a *feasible* timed path. Let $\mathcal{P}$ denote the set of all feasible timed paths.

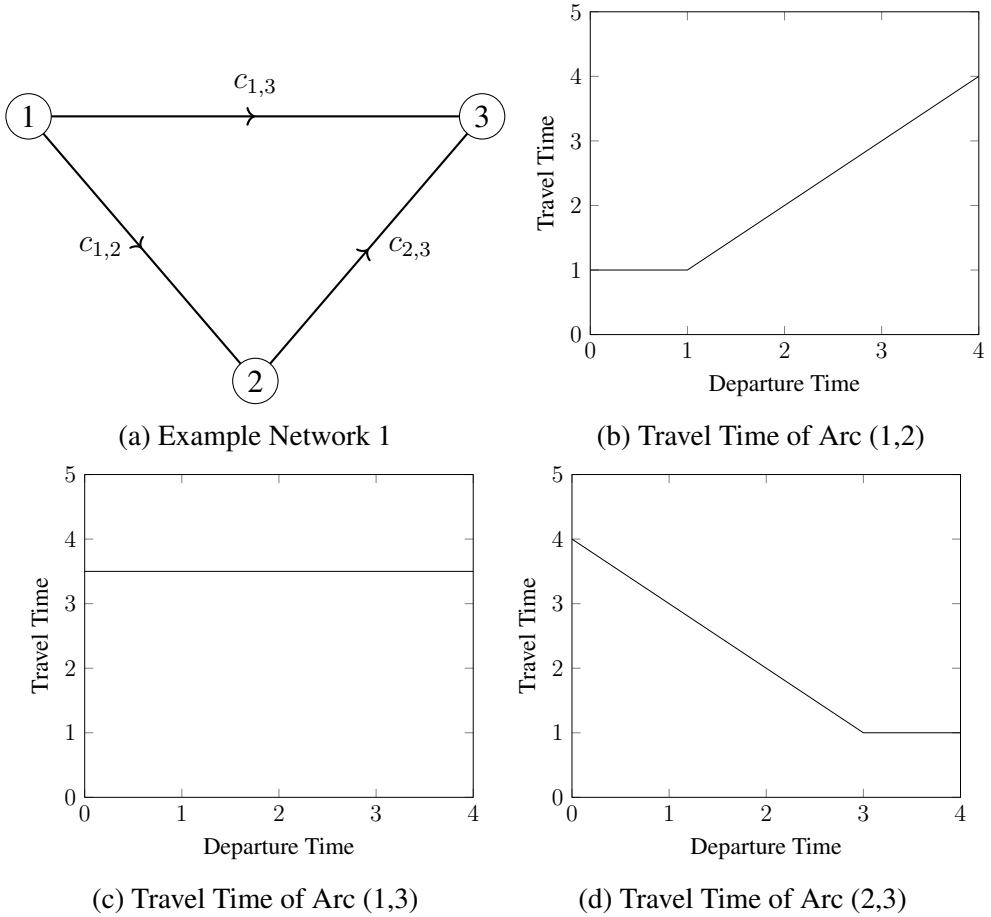
Three objectives are of interest:

1. minimizing the arrival time at the destination, $\min_{P \in \mathcal{P}} t_{m(P)}^P$,
2. minimizing the duration of the path from origin to destination, $\min_{P \in \mathcal{P}} (t_{m(P)}^P - t_1^P)$,
3. minimizing the travel time on the path from origin to destination, $\min_{P \in \mathcal{P}} \sum_{k=1}^{m(P)-1} c_{i_k^P, i_{k+1}^P}(t_k^P)$.

In the rest of the chapter, for notational convenience, we drop the P in superscripts.

To illustrate these three objectives, consider the network and arc travel time functions shown in Figure 2.1, with $n = 3$ and given time horizon $[0, 4]$. The timed path minimizing arrival time at the destination is $((1, 0), (3, 3.5))$: the path would traverse arc $(1, 3)$, starting at $t = 0$ and arriving at $t = 3.5$, giving an objective value of 3.5. The timed path of minimum duration is $((1, 1.5), (2, 3), (3, 4))$: the path would traverse arc $(1, 2)$, departing at $t = 1.5$ and arriving at $t = 3$, and then traverse arc $(2, 3)$, departing at $t = 3$ and arriving at $t = 4$, giving an objective value of 2.5. As there was no waiting on this path, it also has a

travel time of 2.5. However, this is not the minimum travel time. There are multiple optimal solutions for the problem of minimizing the travel time. For any $t_1 \in [0, 1]$, the timed path $((1, t_1), (2, 3), (3, 4))$ has minimum travel time: it would traverse arc $(1, 2)$, departing at t_1 and arriving at $t_1 + 1 \in [1, 2]$, then wait at node 2 until $t = 3$, and then traverse arc $(2, 3)$, departing at $t = 3$ and arriving at $t = 4$, giving an objective value of 2.



(a) Example Network 1

(b) Travel Time of Arc (1,2)

(c) Travel Time of Arc (1,3)

(d) Travel Time of Arc (2,3)

Figure 2.1: Example to illustrate the differences between the three objectives.

Note that because the travel time functions on the arcs have the FIFO property, when minimizing the arrival time at the destination or the duration of the path from origin to destination, an optimal path will have $t_k + c_{i_k,i_{k+1}}(t_k) = t_{k+1}$ for $k = 1, \ldots, m(P) - 1$. In other words, there is no benefit in waiting at intermediate nodes and there must be an optimal solution that is waiting-free. Furthermore, when minimizing the arrival time at the destination, an optimal path will have $t_1^P = 0$.

Finding a timed path that reaches the destination as early as possible, given a fixed start time at the origin, can be done, in polynomial time, with straightforward extensions of algorithms for the standard shortest path problem (Cooke and Halsey 1966, Orda and Rom 1990, Dean 2004b). Finding a timed path that departs the origin as late as possible, given a fixed end time at the destination, can be solved similarly. This is done by pre-computing *reverse* travel time functions: given an arc (i, j) and an arrival time t, the reverse function for (i, j) evaluated at t gives the travel time τ so that starting traversal of (i, j) at time $t - \tau$ results in arrival at j at time t. In what follows, we refer to a problem having a given, fixed, start or end time, as a *Time-Dependent Shortest Path Problem* (TDSPP), and an optimal solution to the problem as a *Time-Dependent Shortest Path* (TDSP). As is the case in finding standard shortest paths, algorithms solving a TDSPP can just as easily provide either the forwards or the backwards shortest path *tree*. Solving a TDSPP is a key subroutine in algorithms for more complex settings.

The MDP has attracted much attention since the early work of Orda and Rom (1990). There are two classes of approach: discrete and continuous. In discrete approaches, such as that of Chabini (1998), time is discretized. Travel along an arc may only start at a time point in the discretization and the arrival time at the end of the arc is mapped, in some way, to a time point in the discretization. In other words, the travel time on an arc is forced to conform to the time discretization. Discrete approaches are thus inexact and rely heavily on the quality of the discretization. A denser discretization leads to a better approximation, but an increase in computation time. Continuous methods, such as the Dijkstra's algorithm variants in Orda and Rom (1990), Nachtigall (1995), Ding et al. (2008), and the A* algorithm variant in Kanoulas et al. (2006), create and update arrival time functions at each node, and are exact. The *arrival time function* at a node takes as input the time of departure at the origin and gives the earliest time of arrival at the node for any path departing from the origin at the given time. In other words, the arrival time function at a node gives the value of the TDSPP with this node as the destination, for every

possible start time at the origin. The complexity analysis of these methods is given in terms of operations that store and manipulate arrival time functions. However, the analysis does not give complexity of these operations in terms of the problem parameters.

Even for continuous piecewise linear functions, it was only recently that an exact algorithm that is polynomial in the total number of breakpoints (in the piecewise linear functions) was proposed (Foschini et al. 2014). Furthermore, as far as we are aware, this is the *only* such algorithm. The authors prove the intuitive result that there is an optimal path that either starts its traversal of some arc exactly at a breakpoint of the arc's travel time function, or starts at node 1 at time $t = 0$ or ends at node n at time $t = T$. Their algorithm investigates all arcs (i, j) and all breakpoints, t, of the function $c_{i,j}$, solves the TDSPP from i to n starting at time t and the TDSPP from 1 to i ending at time t. Concatenating the two resulting paths yields a feasible timed path, and one such path must be optimal. Foschini et al. (2014) observe that this algorithm has computational complexity $\mathcal{O}(K^{\text{arcs}} \times SP)$, where SP is the complexity of solving a time-dependent shortest path problem (with a fixed starting time) in the given network and $K^{\text{arcs}} = \sum_{(i,j)\in A} k_{i,j}$ is the total number of breakpoints, where $k_{i,j}$ is the number of breakpoints in the function $c_{i,j}(\cdot)$, for each $(i, j) \in A$. We note here that, in fact, if arcs with the same tail node have common breakpoints in their travel time functions, the complexity is $\mathcal{O}(K^{\text{nodes}} \times SP)$ for $K^{\text{nodes}} < K^{\text{arcs}}$ given by

$$K^{\text{nodes}} = \sum_{i\in N} \left| \bigcup_{j:\ (i,j)\in A} B_{i,j} \right|,$$

where $B_{i,j}$ is the set of breakpoints of $c_{i,j}(t)$ over $t \in [0, T]$. It is then natural to consider breakpoints as associated with nodes, where if the travel time function of arc (i, j) has a breakpoint at time t, we say that *node i has a breakpoint at time t*. We may also say that (i, t) *is a breakpoint*. In addition, we shall consider $(1, 0)$ and (n, T) to also be breakpoints. While the result is intuitive, it is surprising, since Dean (2004b) proposed that the minimum arrival time function has a superpolynomial number of linear pieces. This is due to arbitrary

fractional departure times being possible even with fully integer data.

As we show in Section 2.4, the ideas of Foschini et al. (2014) can be extended to the MTTP, and used to define an algorithm for the MTTP that is polynomial in the total number of breakpoints. This is in contrast to the recent work presented in a research report of Omer and Poss (2019b), who show that in the presence of a constraint on the waiting time at nodes, the problem is NP-hard.

In this chapter, we develop dynamic discretization discovery algorithms for MDP and MTTP that investigate only a very small fraction of the total number of breakpoints, while solving the problem to proven optimality.

2.3 A Dynamic Discretization Discovery Algorithm for the MDP

For ease of exposition, this chapter focuses only on the case that the FIFO property holds *strictly*. Only minor modifications to the algorithm and proof of correctness are needed in the case the FIFO property is not strict.

We first provide some preliminary definitions, illustrated using the instance given by the network in Figure 2.2 with the travel time functions whose graphs are shown in Figure 2.4. The time interval is $[0, 5]$, breakpoints for each arc are at every integer point, with the exception of arc $(3, 4)$ which only has breakpoints at $0, 1, 2$ and 5. The arc travel time functions and their reverse functions are stated in Appendix A.

Given a time $t \in [0, T]$ that represents a specific arrival time at node n, a corresponding *Backwards Shortest Path Tree* (BSPT) is a *Time-Expanded Network* (TEN), denoted by $\mathcal{B}^t$, which is an in-tree rooted at (n, t) defined by a set of timed nodes of the form $(i, t_i) \in N \times (-\infty, t]$ and timed arcs of the form $((i, t_i), (j, t_j))$ satisfying:

- for each $i \in N$ such that n is reachable from i, there is exactly one time, t_i, for which the timed node (i, t_i) is in $\mathcal{B}^t$,
- t_i is the latest departure time from i that allows arrival at n at time t,

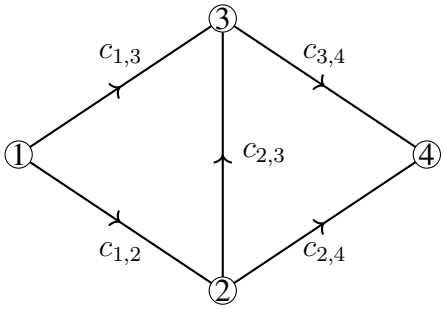
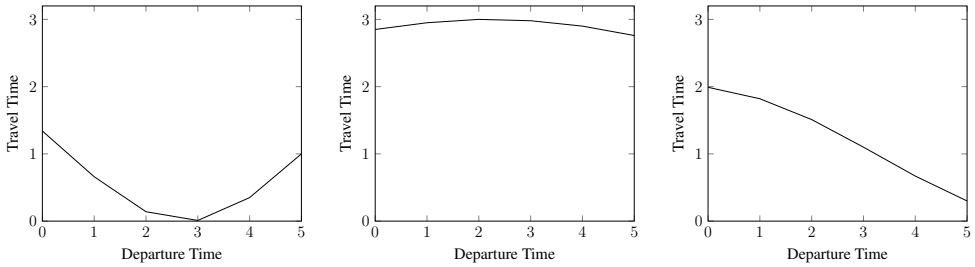


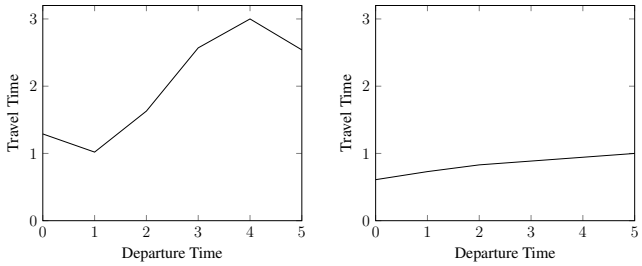
Figure 2.2: Example Network 1

Table. 2.3: Arc Travel Times at Each BP

BP Time	Arc Travel Times				
	(1,2)	(1,3)	(2,3)	(2,4)	(3,4)
0	1.34	2.85	1.99	1.29	0.61
1	0.66	2.95	1.82	1.02	0.73
2	0.14	3.00	1.51	1.63	0.83
3	0.01	2.98	1.10	2.57	—
4	0.35	2.90	0.67	3.00	—
5	1.00	2.76	0.30	2.54	1.00



(a) Travel Time of Arc (1,2) (b) Travel Time of Arc (1,3) (c) Travel Time of Arc (2,3)



(d) Travel Time of Arc (2,4) (e) Travel Time of Arc (3,4)

Figure 2.4: Arc Travel Times

- $(i, j) \in A$,
- $t_i + c_{i,j}(t_i) = t_j$, and
- there is a unique path from (i, t_i) to (n, t) in $\mathcal{B}^t$, which solves the TDSPP from origin i starting at time t_i to destination n.

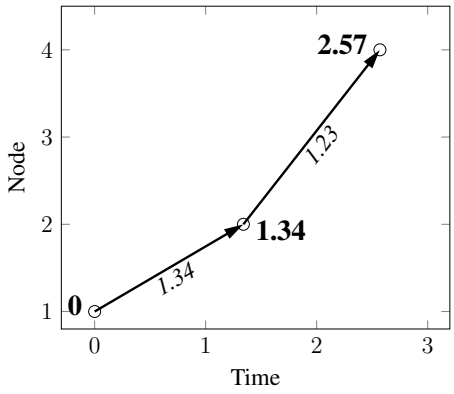
As mentioned in the previous section, the TDSPP can be solved, and hence a BSPT found, easily, using straightforward adaptations of standard shortest path algorithms.

The BSPT for node 4 at $t = 2.57$ in the example is shown in Figure 2.5b. It is the TEN with nodes $\{(1, 0.00), (2, 1.34), (3, 1.76), (4, 2.57)\}$, (times are accurate to two decimal places), and arcs given by the unique timed copies of the arcs $(1, 2)$, $(2, 4)$ and $(3, 4)$ induced by the timed nodes.

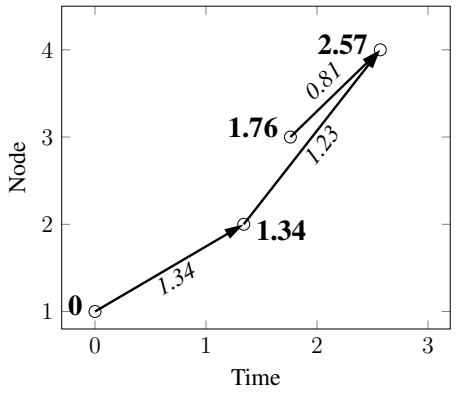
Note that for a given $t \in [0, T]$, in constructing $\mathcal{B}^t$, it may be that for some i it is impossible to reach n from i by time t for any positive departure time at i. Of course, if this is the case for $t = T$, the node i can simply be eliminated from the network in a preprocessing step. Otherwise, for simplicity of exposition in what follows, we ensure that a BSPT includes a timed node for every $i \in N$ by extending $c_{i,j}(t)$ to $t < 0$: we set $c_{i,j}(t) = c_{i,j}(0)$ for all $t < 0$ and all $(i, j) \in A$ and permit timed nodes of the form (i, t_i) with $t_i < 0$ to be included in $\mathcal{B}^t$.

Our dynamic discretization discovery algorithm is based on the following key observations about BSPTs. First, it is clear from the FIFO property that if (i, s) is a node in BSPT $\mathcal{B}^t$ and (i, s') is a node in BSPT $\mathcal{B}^{t'}$ with $t' > t$, then $s' > s$. Second, any minimum duration timed path from 1 to n that arrives at node n at time t or later has a *representative* timed node sequence in $\mathcal{B}^t$, meaning that for each node-time pair (i, s) in the path, there is a timed node (i, s') in $\mathcal{B}^t$, which is not later: $s' \leq s$. Note that such a representative timed node sequence in $\mathcal{B}^t$ need not follow timed arcs in $\mathcal{B}^t$. This motivates the following definition.

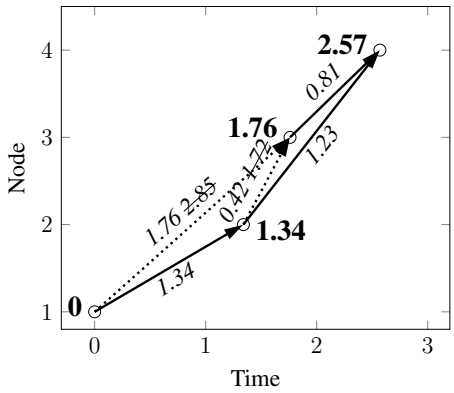
An *Arc-completed Backwards Shortest Path Tree* (ABSPT) is the TEN formed from a



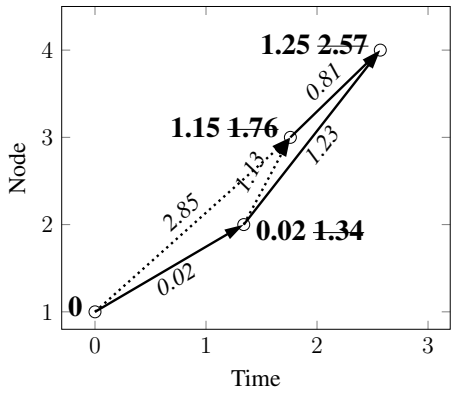
(a) SP from $(1, 0)$ (times of timed nodes are bold, travel times are italicized)



(b) BSPT from $(4, 2.57)$ (times of timed nodes are bold, travel times are italicized)



(c) ABSPT $\overline{B}^{2.57}$ (actual travel times are strike-through, times of timed nodes are bold, travel times implied by head and tail node times are italicized)



(d) ABSPT $\overline{B}^{2.57}$ with UTT calculated with respect to $\mathcal{B}^5$ are italicized, times of timed nodes are strike-through and shortest path node labels are bold

Figure 2.5: Procedure to generate the ABSPT corresponding to $(1, 0)$.

given BSPT by adding timed arcs, $((i,t_i),(j,t_j))$, for every $(i,j) \in A$ having both (i,t_i) and (j,t_j) in the BSPT. By the definition of a BSPT, it must be that $t_i + c_{i,j}(t_i) \geq t_j$ for all such arcs in the ABSPT. In other words, $t_j - t_i \leq c_{i,j}(t_i)$ for all $((i,t_i),(j,t_j))$ in an ABSPT. The ABSPT for the example of $\mathcal{B}^t$ with $t = 2.57$ is shown in Figure 2.5c, with the values of $c_{i,j}(t_i)$ crossed out, and replaced by the value $t_j - t_i$, on all new arcs added to form the ABSPT. We use the notation $\overline{\mathcal{B}}^t$ to denote the ABSPT formed from $\mathcal{B}^t$.

Our algorithm works by maintaining a *list* of ABSPTs, ordered by their time at the end node. The list is initialized with two ABSPTs: the ABSPT for the earliest time that the end node can be reached and the ABSPT with end time given by the end of the time horizon. These two times define the time interval of interest: all feasible timed paths must arrive at the end node within this interval. Additional ABSPTs are generated dynamically, subdividing this interval. In the example, the initial ABSPTs are $\overline{\mathcal{B}}^{2.57}$ and $\overline{\mathcal{B}}^5$ (since $T = 5$). The latter has timed nodes $(1, 2.90)$, $(2, 2.92)$, $(3, 4.05)$ and $(4, 5)$.

The algorithm relies on the following third, and final, key observation, which concerns two ABSPTs that appear *consecutively* in the list. Say $\overline{\mathcal{B}}^t$ and $\overline{\mathcal{B}}^{t^+}$, whose end times t and t^+, respectively, satisfy $t^+ > t$, are two such ABSPTs. For each arc $((i,s_i),(j,s_j))$ in the earlier ABSPT, $\overline{\mathcal{B}}^t$, we define its *Underestimated Travel Time* (UTT) with respect to $\overline{\mathcal{B}}^{t^+}$ by

$$\underline{c}_{(i,s_i),(j,s_j)} = \min_{\tau} \{c_{ij}(\tau) \mid s_i \leq \tau \leq s_i^+\},$$

where (i, s_i^+) is the timed node for i in the later ABSPT. Figure 2.5d shows the UTT for the example with $t = 2.57$ and $t^+ = 5$ along each arc. These are calculated over the time intervals at each node between the times for that node in $\overline{\mathcal{B}}^{2.5}$ and $\overline{\mathcal{B}}^5$. For example, the UTT for timed arc $((1,0),(2,1.34))$ is the minimum of $c_{1,2}(\tau)$ over τ in the interval $[0, 2.90]$, which is 0.02, achieved at the right-hand end of the interval. By contrast, the UTT for timed arc $((1,0),(3,1.76))$ is the minimum of $c_{1,3}(\tau)$ over τ in the interval $[0, 4.05]$, which is 2.85, achieved at the left-hand end of the interval.

The shortest path with respect to the UTT in each ABSPT in the list provides a lower

bound on the duration of any feasible timed path arriving in the subinterval between its arrival time at the end node, and that of the next ABSPT in the list. For the example, Figure 2.5d shows the node labels associated with the least UTT path in the ABSPT $\overline{\mathcal{B}}^{2.57}$ against each node. Since the node label on the end node is 1.25, no feasible path arriving at the end node between times 2.57 and 5 can have duration less than 1.25.

The least such lower bound, over all ABSPTs in the list, is a lower bound on the MDP. In the example, since 2.57 is the earliest time that the end node can be reached and 5 is the end of the time horizon, 1.25 is, in fact a lower bound on the duration of *any* feasible timed path.

The lower bound resulting from the list of ABSPTs can be improved by refining the discretization: the insertion of a new ABSPT with end time in the interval following the end time of the ABSPT that gave the lower bound will increase (or at least not decrease) its UTT.

An upper bound can always be obtained from an ABSPT: if $(1, t_1)$ and (n, t_n) are the two timed nodes in the ABSPT on the start and end node respectively, then $t_n - t_1$ must be an upper bound, by the definition of the BSPT it was generated from. The algorithm maintains the best such upper bound. In the example, the ABSPT $\overline{\mathcal{B}}^{2.57}$ has duration 2.57 and the ABSPT $\overline{\mathcal{B}}^{5}$ has duration 2.10. Clearly the latter gives the best of these upper bounds. So at this stage of the example, the two ABSPTs give a best lower bound of 1.25 and a best upper bound of 2.10.

The above elements alone – obtaining lower and upper bounds from the list of ABSPTs, and inserting a new ABSPT in the subinterval following the end time of the ABSPT that gave the lower bound – can give an algorithm that converges to the optimal solution, for example, by choosing the new ABSPT to have end time bisecting the subinterval. However, since the optimal starting time is often a fractional time, the algorithm will theoretically not terminate with the optimal solution (computationally, it will terminate within a set tolerance). By exploiting the observation of Foschini et al. (2014) – that there is an

optimal solution using some arc travel time function breakpoint – the end times for new ABSPTs can be created more carefully, and subintervals eliminated from further consideration, without the need for new ABSPTs. The result is a finitely terminating algorithm. We now describe how we make use of breakpoints in the arc travel time functions to arrive at such an algorithm.

First, we only create ABSPTs that contain at least one node-time pair (i, t) that is a breakpoint. Clearly the initial two ABSPTs satisfy this requirement, as $(1, 0)$ is in the first ABSPT and (n, T) is in the second, and both $(1, 0)$ and (n, T) are deemed to be breakpoints.

Now suppose two consecutive ABSPTs in the list have timed nodes denoted by (i, t_i) for the earlier ABSPT, and (i, t_i^+) for the later, and assume that the earlier ABSPT gave the current lower bound. First consider the case that for some arc $(i, j) \in A$, its arc travel time function has a breakpoint at τ_i with $t_i < \tau_i < t_i^+$. In this case, we say that the breakpoint (i, τ_i) lies *between* the two ABSPTs, and a new ABSPT containing the breakpoint can be constructed by first finding a *Shortest Path* (SP) to n, starting from i at time τ_i. Say this SP arrives at n at time τ_n. Then the BSPT from τ_n, $\mathcal{B}^{\tau_n}$, must include the timed node (i, τ_i). We call its arc completion, $\overline{\mathcal{B}}^{\tau_n}$, the ABSPT *corresponding to* (i, τ_i). By properties of ABSPTs discussed earlier, it must be that $t_n < \tau_n < t_n^+$, and the ABSPT corresponding to (i, τ_i) can be inserted in the list between the two ABSPTs with end times t_n and t_n^+.

Now consider the remaining case, that for every arc $(i, j) \in A$, its arc travel time function has no breakpoint between t_i and t_i^+. In this case, we exploit the fact that the arrival time function at n for departures from node 1 between t_1 and t_1^+ is concave if no minimum duration path that departs between t_1 and t_1^+ contains a breakpoint (Foschini et al. 2014). This situation occurs when there are no more breakpoints left to explore between two ABSPTs, i.e., in this remaining case. When this happens, concavity of the arrival time function at n implies that the minimum duration path departing between t_1 and t_1^+ departs at either t_1 or t_1^+, both of which have already been calculated as the upper bound from the respective ABSPTs. Hence the lower bound for the earlier ABSPT can be updated to one

of these upper bounds. In this case, we say the status of the ABSPT is *resolved*.

A high-level overview of our algorithm can be found in Algorithm 1. For a given ABSPT in the ordered list maintained by the algorithm, $\overline{\mathcal{B}}^t$, say, having timed nodes denoted by (j, t_j) for each j, the procedure $computeUB(\overline{\mathcal{B}}^t)$ simply returns $t_n - t_1$. The procedure $computeLB(\overline{\mathcal{B}}^t)$ constructs the UTT for each arc in the ABSPT with respect to the subsequent ABSPT in the list, and then finds the least UTT path from $(1, t_1)$ to (n, t_n) in the ABSPT, returning its value. The values computed by these procedures are recorded, and associated with the ABSPT, using the notation UB^t and LB^t, respectively. Note that the UTT for the last ABSPT in the list, $\overline{\mathcal{B}}^T$, are taken to be the actual travel times if the arc is in the BSPT, and infinity otherwise, since no later departure time at any node can reach the end node within the time horizon. Thus the lower bound from this ABSPT is identical to its upper bound.

Algorithm 1 is actually a family of algorithms, since it does not specify which breakpoint to choose when there are multiple breakpoints between $\overline{\mathcal{B}}^t$ and $\overline{\mathcal{B}}^{t^+}$. We discuss alternative schemes for breakpoint selection, and their effects on computational performance, in Subsection 2.5.2.

We illustrate the algorithm on the example from Figure 2.2 and Figure 2.3: see Figure 2.6. The algorithm's ABSPT list is initialized with $\overline{\mathcal{B}}^{2.57}$, which corresponds to $(1, 0)$, (the SP starting from $(1, 0)$ arrives at node 4 at time $t_0 = 2.57$), and $\overline{\mathcal{B}}^5$, as discussed earlier. Recall that the former gives the best lower bound on the duration, $LB = 1.25$, and the latter gives the best upper bound, $UB = 2.10$. In the first iteration, since arc $(1, 2)$ has a breakpoint at time $\tau = 1 \in [0, 2.90]$, the subinterval arising from timed nodes $(1, 0)$ in $\overline{\mathcal{B}}^{2.57}$ and $(1, 2.90)$ in $\overline{\mathcal{B}}^5$ gives the lower bound, a new breakpoint, $(j, \tau) = (1, 1)$ between the two ABSPTs is found. The SP starting from $(1, 1)$ is $((1, 1), (2, 1.66), (4, 3.0826))$, which reaches node 4 at time 3.08 (to two decimal places), and so $\overline{\mathcal{B}}^{3.08}$ is created and added to the list. The list now has three ABSPTs, as shown in Figure 2.6b. Computing its upper and lower bounds yields $UB^{3.08} = 2.08$ and $LB^{3.08} = 1.45$ (to two decimal places). After

input : digraph $D = (N, A)$, latest time, T, arc travel time function $c_{i,j}(t)$ for all $t \in [0, T]$, each $(i, j) \in A$
output: minimum duration path from node 1 to n departing and arriving at times in $[0, T]$
Solve a SP starting from $(1, 0)$ to determine t_0, the earliest time that n can be reached ;
Initialize ordered list of ABSPTs: set $L \leftarrow (\overline{\mathcal{B}}^{t_0}, \overline{\mathcal{B}}^{T})$;
$UB \leftarrow \min\{computeUB(\overline{\mathcal{B}}^{t_0}), computeUB(\overline{\mathcal{B}}^{T})\}$;
$LB^{t_0} \leftarrow computeLB(\overline{\mathcal{B}}^{t_0})$;
Set $LB \leftarrow LB^{t_0}$, set $t \leftarrow t_0$ and set $t^+ \leftarrow T$;
while $(LB < UB)$ **do**
  if *some breakpoint* (j, τ) *lies between* $\overline{\mathcal{B}}^{t}$ *and* $\overline{\mathcal{B}}^{t^+}$ **then**
    Solve the SP starting from (j, τ) to determine, s, the earliest arrival at n ;
    Create the new ABSPT $\overline{\mathcal{B}}^{s}$ and set $UB^s \leftarrow computeUB(\overline{\mathcal{B}}^{s})$;
    if $UB^s < UB$ **then**
      $UB \leftarrow UB^s$
    end
    Insert $\overline{\mathcal{B}}^{s}$ in the list L between $\overline{\mathcal{B}}^{t}$ and $\overline{\mathcal{B}}^{t^+}$;
    $LB^t \leftarrow computeLB(\overline{\mathcal{B}}^{t})$;
    $LB^s \leftarrow computeLB(\overline{\mathcal{B}}^{s})$;
  else
    The status of $\overline{\mathcal{B}}^{t}$ is resolved: set $LB^t \leftarrow UB^t$;
  end
  Update the lower bound: set $t \leftarrow \arg\min_{\tau} LB^{\tau}$ and $LB \leftarrow LB^t$;
  Identify the next ABSPT in the list: $t^+ \leftarrow \min\{\tau : \overline{\mathcal{B}}^{\tau} \text{ is in } L \text{ and } \tau > t\}$;
end

Algorithm 1: *Dynamic Discretization Discovery* (DDD) Algorithm for the MDP

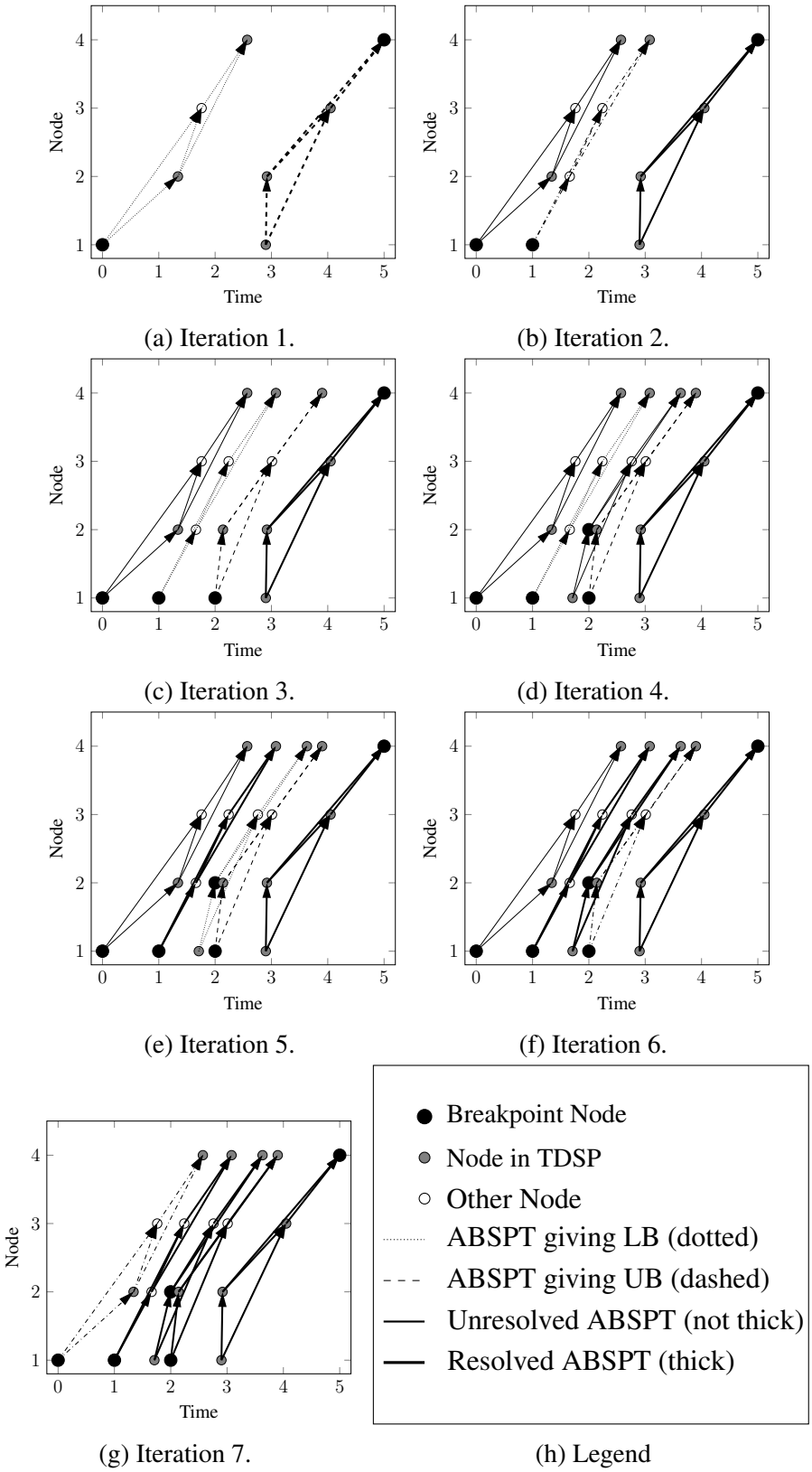
updating the UTT and recomputing the lower bound for $\overline{\mathcal{B}}^{2.57}$, we update $LB^{2.57} = 1.89$. Now $UB = 2.08$ and $LB = 1.45$. Since $LB < UB$, we continue with the algorithm. We proceed to add the ABSPTs corresponding to $(1, 2)$ and $(2, 2)$ ($\overline{\mathcal{B}}^{3.89}$ and $\overline{\mathcal{B}}^{3.63}$ respectively). The latter is the third ABSPT in the list at Iteration 4. In Iteration 4, $\overline{\mathcal{B}}^{3.08}$ gives the current lower bound, $LB = 1.71$, and is the second ABSPT in the list. There are no breakpoints between the second and third ABSPTs in the list, so we replace the lower bound of $\overline{\mathcal{B}}^{3.08}$ with its upper bound: $LB^{3.08} \leftarrow 2.08$. The new lower bound is given by the ABSPT corresponding to $(2, 2)$, $\overline{\mathcal{B}}^{3.63}$, so at Iteration 5, $LB = 1.89$. We proceed to replace

the lower bound of $\overline{\mathcal{B}}^{3.63}$ and $\overline{\mathcal{B}}^{3.89}$ with their upper bound due to the lack of breakpoints between them and the next ABSPT in the list, at which stage $UB = 1.90$, which is equal to $LB = 1.90$ and hence the algorithm terminates. The optimal path is the one that corresponds to the upper bound, which originates from ABSPT $\overline{\mathcal{B}}^{3.89}$ and gives us the minimum duration path $((1, 2.00), (2, 2.14), (4, 3.89))$.

2.4 A Dynamic Discretization Discovery Algorithm for the MTTP

In this section, we extend the ideas of the dynamic discretization discovery algorithm for the MDP to a dynamic discretization discovery algorithm for the MTTP. However, we first observe that the list L of ABSPTs used in Algorithm 1 can be reinterpreted as a single TEN formed by the union of timed nodes and timed arcs in its ABSPTs. The single TEN should also include all timed arcs of the form $((1, t), (1, t^+))$ for *consecutive*, meaning chronologically adjacent, timed nodes $(1, t)$ and $(1, t^+)$ with $t < t^+$. Thus the TEN includes timed arc $((1, t), (1, t^+))$ only if no ABSPT in L includes timed node $(1, t'')$ with $t < t'' < t^+$. Similarly, it should include all timed arcs of the form $((n, t), (n, t^+))$ for consecutive (n, t) and (n, t^+) with $t < t^+$. These arcs model waiting at the origin for the right time to start the path and waiting at the destination after the end of the path. By assigning all such arcs a UTT of zero, the lower bound calculated at each iteration of the algorithm is precisely the value of the lower bound timed path from $(1, 0)$ to (n, T) in this single TEN. Algorithm 1 exploits the following observations: (i) finding a lower bound timed path in this single TEN can be decomposed into lower bound timed path calculations for each individual ABSPT, and (ii) these calculations will change for only one ABSPT when a new ABSPT is added.

Recall that the key difference between the MDP and the MTTP is that in the latter, it may be that an optimal path must wait at a node other than the origin. This prevents decomposition into individual ABSPTs; our lower bound for the MTTP will be calculated using a single TEN that includes *waiting arcs* between timed nodes at the same node in N,



(a) Iteration 1. (b) Iteration 2.

(c) Iteration 3. (d) Iteration 4.

(e) Iteration 5. (f) Iteration 6.

(g) Iteration 7. (h) Legend

Figure 2.6: TEN in each iteration for the example in Figure 2.2 (note that an ABSPT may satisfy multiple criteria in the legend).

which accommodate the option to wait at that node. In addition to loss of the decomposition property when solving the MTTP, the ability to use ABSPTs as the basis for the TEN is also lost: ABSPTs, alone, are not sufficient to characterize solutions to the MTTP. Instead, the TEN we construct is based on unions of *forwards* and backwards trees from a node *other than* the origin or destination, starting and ending, respectively, at a given time. In what follows, we give the motivation and details for this construction.

2.4.1 Foundational Theory for the MTTP

The foundations for our algorithm for the MTTP start with the simple observation that any solution to a MTTP instance will contain periods of travel and periods of waiting, as shown, for example, in Figure 2.7. Indeed, any timed path corresponds to a sequence of waiting-free timed paths.

Definition 8. *A* travel subpath *of a given timed path is created from a maximal sequence of consecutive timed nodes in the timed path, say* $(i_1, t_1), \ldots, (i_k, t_k)$ *is such a sequence, satisfying (i)* $t_{\ell+1} = t_\ell + c_{i_\ell, i_{\ell+1}}(t_\ell)$ *for all* $\ell = 1, \ldots, k-2$ *and (ii) either* $t_k > t_{k-1} + c_{i_{k-1}, i_k}(t_{k-1})$*, which indicates that the path waits at* i_k*, or* $i_k = n$ *and* $t_k = t_{k-1} + c_{i_{k-1}, i_k}(t_{k-1})$*. The travel subpath is then defined as the timed node sequence* $((i_1, t_1), \ldots, (i_{k-1}, t_{k-1}), (i_k, \tau))$ *where* $\tau = t_{k-1} + c_{i_{k-1}, i_k}(t_{k-1})$.

For example, if $((1, 0), (2, 6.9), (3, 7.9), (4, 8.1), (5, 8.5))$ is the sequence of nodes in a timed path, with $c_{12}(0) = 1$, $c_{23}(6.9) = 0.2$, $c_{34}(7.9) = 0.2$ and $c_{45}(8.1) = 0.4$, then the timed path has three travel subpaths, given by the timed node sequences $((1, 0), (2, 1))$, $((2, 6.9), (3, 7.1))$ and $((3, 7.9), (4, 8.1), (5, 8.5))$. A timed path like this is illustrated in Figure 2.7, which shows a TEN for the network $D = (\{1, 2, 3, 4, 5\}, \{(1, 2), (2, 3), (3, 4), (4, 5)\})$, with timed path shown. Its three travel subpaths are shown in thick black.

A travel subpath of a timed path is maximally waiting-free: it cannot be extended at either end while remaining waiting-free. A timed path can be viewed as a sequence of

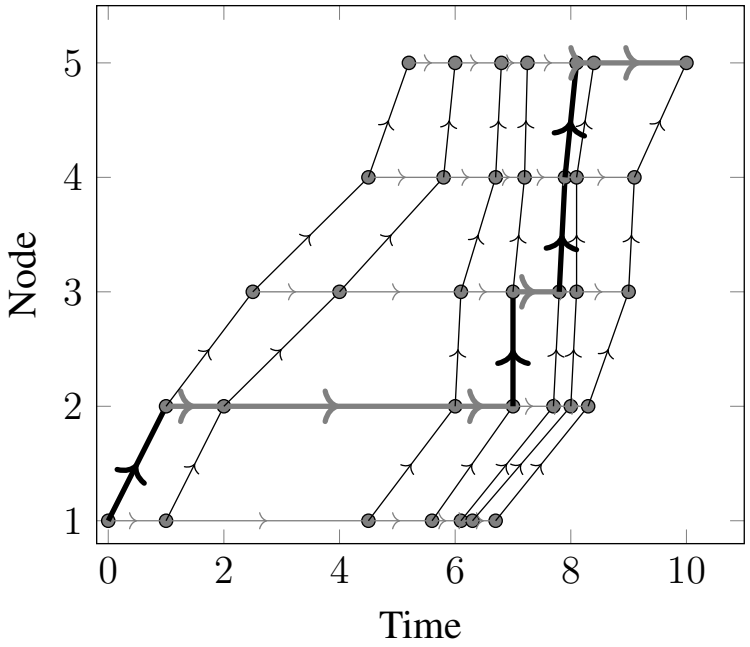


Figure 2.7: Illustration of a timed path with 3 periods of waiting in thick gray and 3 travel subpaths in thick black, in the TEN for a network consisting only of the arcs $(1, 2), (2, 3), (3, 4)$ and $(4, 5)$.

travel subpaths, with waiting between them: whenever timed nodes (i, t_i) and (j, t_j) appear consecutively in a timed path and have $t_j > t_i + c_{i,j}(t_i)$, then the timed node $(j, t_i + c_{i,j}(t_i))$ is the end of one of its travel subpaths and the timed node (j, t_j) is the start of the next. Note that a travel subpath is a timed path in its own right, and is, by construction, waiting-free.

Obviously any sequence of timed paths with the property that $i = j$ and $t_j \geq t_i$ whenever (i, t_i) is the last timed node in one of them and (j, t_j) is the first timed node of the next can be converted to a single timed path simply by omitting the last timed node of every timed path in the sequence, except the last, and then concatenating them. Clearly, for any timed path P there is a unique sequence of waiting-free timed paths that can be used to generate P by this process, the elements of which are precisely its travel subpaths. We say that this sequence *corresponds to* P and *vice versa*.

We now present the key lemma needed to extend the observation of Foschini et al. (2014) to the minimum travel time case.

Lemma 1. *Given an optimal solution, P, to an MTTP instance on network D with arc*

travel time functions c and time horizon $[0, T]$, consider any travel subpath of it, S say. Say S starts at timed node (i, t_i) and ends at timed node (j, t_j). Then S is an optimal solution to the MDP instance on the same network, D, with the same arc travel time functions, c, but with origin i, destination j and time horizon $[\tau^-, \tau^+]$, where $\tau^- \leq t_i$ is the end time of the travel subpath immediately preceding S in P, if any, and $\tau^+ \geq t_j$ is the start time of the travel subpath immediately following S in P, if any. Here $\tau^- = 0$ if $i = 1$ (there is no travel subpath preceding S) and $\tau^+ = T$ if $j = n$ (there is no travel subpath after S).

Proof. Given P, S, i, j, t_i, t_j, τ^- and τ^+ as defined above for the MTTP instance, we will refer to the MDP instance with origin i, destination j and time horizon $[\tau^-, \tau^+]$ as the $i \to j$ *MDP instance*. Observe that S is a feasible solution to the $i \to j$ MDP instance and its total travel time is given by $t_j - t_i$, which is precisely its duration, since S is a travel subpath and hence is waiting-free. Thus S has total travel time *at least* the optimal value of the $i \to j$ MDP instance. Also observe that the duration of any timed path is at least its total travel time. Now suppose that S is *not* optimal for the $i \to j$ MDP instance. Then it has strictly greater total travel time than the optimal value of the $i \to j$ MDP instance, which is at least the total travel time of any of its optimal solutions. Hence, S can be replaced in the sequence of waiting-free timed paths that corresponds to P by any solution to the $i \to j$ MDP instance. The resulting, new, sequence of waiting-free timed paths must yield a corresponding timed path that is feasible for the MTTP and has strictly lower total travel time than that of P, contradicting the optimality of P. □

The following is the extension of the observation of Foschini et al. (2014) to the MTTP.

Proposition 1. *For any MTTP instance, there exists an optimal timed path such that each of its travel subpaths includes at least one timed node at an arc travel time function breakpoint.*

Proof. Suppose that P is an optimal timed path for a MTTP instance having a travel subpath, S, that does *not* include a timed node at an arc travel time function breakpoint. Then

by Lemma 1, S is an optimal solution to the MDP instance on the same network with the same arc travel time functions, but with origin i, destination j and time horizon $[\tau^-, \tau^+]$, where i, j, τ^- and τ^+ are defined as in the statement of the lemma. By the results of Foschini et al. (2014), there must be another optimal solution to this MDP instance, S' say, which includes at least one timed node at a breakpoint, or which starts at i at time τ^- or ends at j at time τ^+. Since S and S' are both waiting-free, their total travel time is precisely their durations, which are identical, as both are optimal for the same MDP instance. By replacing S with S' in the sequence of waiting-free timed paths corresponding to P, a corresponding new timed path, P', is obtained. Clearly P' is feasible for the MTTP, has the same total travel time as that of P, and has one fewer travel subpaths that does not include a timed node at a breakpoint. (In the case that S' starts at time τ^-, and (i, τ^-) is not at an arc travel time function breakpoint, the travel subpath preceding S is no longer maximal: it now extends to S', resulting in one fewer travel subpaths in P' than in P. The case that S' ends at time τ^+, and (j, τ^+) is not at a breakpoint is similar: S' is not a travel subpath of P' but will extend to the travel subpath that followed S.) □

The above proposition shows that, given an MTTP instance, there must exist a (finite) TEN and an arc length for each of its arcs so that any shortest path in the TEN corresponds to an optimal solution of the MTTP instance. We define this TEN and its arc lengths below. But first, we extend the definition of a backwards shortest path tree to be one rooted at a timed node (k, t) for *any* node $k \in N$. Such a BSPT is formed by solving a TDSPP to find, for each node, the latest departure time at that node so that a path from the node to k arrives at time t. We denote this BSPT by $\mathcal{B}^{k,t}$. We refer to any such BSPT a *k-BSPT*. Similarly, a *Forwards Shortest Path Tree* (FSPT) is formed by solving a TDSPP to find, for each node, the earliest arrival time at that node on a path from k to that node departing at time t, for any node k. We denote this FSPT by $\mathcal{F}^{k,t}$ and refer to any such FSPT a *k-FSPT*.

Definition 9. *Given a MTTP instance with arc travel time functions c, construct its* Time-Expanded Network with Associated Arc Lengths *(TENL), as follows.*

1. *For each breakpoint (i,t) in the given instance, solve two TDSPP, one to calculate the i-FSPT $\mathcal{F}^{i,t}$ and the other to calculate the i-BSPT $\mathcal{B}^{i,t}$. Form a single TEN from the union of all the timed nodes and timed arcs in $\mathcal{F}^{i,t}$ and $\mathcal{B}^{i,t}$ over all breakpoints, (i,t), in the instance.*

2. *For each node in the given instance, add waiting arcs between chronologically adjacent timed copies of it in the TEN.*

3. *Assign any arc in the resulting TEN, $((i,t),(j,t'))$ say, a* length *of zero if $j = i$ and a length of $c_{i,j}(t)$ (which must be equal to $t' - t$, by construction), otherwise.*

Recall that $(1,0)$ and (n,T) are deemed to be breakpoints in a MTTP instance with node set $\{1,\ldots,n\}$ and time horizon $[0,T]$, so both are nodes in its TENL.

Corollary 1. *Given a MTTP instance with node set $\{1,\ldots,n\}$ and time horizon $[0,T]$, the sequence of timed nodes in any shortest path from $(1,0)$ to (n,T) in its TENL corresponds to an optimal timed path to that MTTP instance.*

Proof. Let P be an optimal solution to the given MTTP instance with the property that each of its travel subpaths includes a breakpoint in the instance. Such a path must exist, by Proposition 1. Suppose P starts with $(1,t_1)$ and ends with (n,t_n). It is not difficult to see that the sequence of travel subpaths corresponding to P induces a path in the TENL from $(1,t_1)$ to (n,t_n) of length equal to the total travel time of P: the latter part of each travel subpath, from the breakpoint it contains to its end, is part of the FSPT for that breakpoint, and the earlier part of each travel subpath, to the breakpoint it contains from its start, is part of the BSPT for that breakpoint. Thus each travel subpath is a path in the TENL, and the sum of the TENL arc lengths over all of the travel subpaths of P is precisely its total travel time. Furthermore, the travel subpaths can obviously be connected via waiting arcs in the TENL, and connected to (n,T) from the end of the last travel subpath and from $(1,0)$ to the start of the first travel subpath, to create a single path in the TENL from $(1,0)$ to (n,T),

while adding nothing to the length. Also obvious is that any timed path in the TENL from $(1, 0)$ to (n, T) has length identical to the sum of lengths of the arcs in same path with all waiting arcs removed, and that the connected components of these form a sequence of waiting-free timed paths whose corresponding timed path departs node 1 no earlier than 0, arrives at n no later than T, and has total travel time equal to the sum of their lengths. Thus every path in the TENL from $(1, 0)$ to (n, T) has length equal to the total travel time of some feasible timed path for the MTTP instance. The result follows. □

Corollary 1 provides a method for solving the MTTP that is analogous to the method of Foschini et al. (2014) for solving the MDP. The difference is that now a shortest path problem over the TENL, in its entirety, must be solved; it cannot be decomposed into distinct shortest path problems for each breakpoint. Since its TENL can be expected to have $\mathcal{O}(K^{\text{nodes}}n)$ nodes and $\mathcal{O}(K^{\text{nodes}}n)$ arcs for an MTTP instance with n nodes and K^{nodes} breakpoints, its size is clearly sensitive to the number of breakpoints in the instance. Specifically, we calculate the following complexity.

Proposition 2. *The complexity of the MTTP on digraph (N, A) having $n = |N|$ nodes and K^{nodes} travel time function breakpoints at nodes is*

$$\mathcal{O}(K^{nodes} \times SSP(n, |A|) + ASPP(K^{nodes}n, K^{nodes}n)),$$

where $SPP(\alpha, \beta)$ denotes the complexity of solving a static shortest path problem on a digraph with α nodes and β arcs, and $ASPP(\alpha, \beta)$ denotes the same thing but for an acyclic digraph.

Proof. For each node, there can be up to two timed copies created by the FSPT and BSPT for each breakpoint, giving up to $2K^{\text{nodes}}$ timed copies of each node and hence at most $2K^{\text{nodes}}$ waiting arcs at each node. Thus there would be a total of at most $2K^{\text{nodes}}n$ nodes in the TENL and at most $2K^{\text{nodes}}n$ waiting arcs. Each breakpoint creates a FSPT, which can have at most n arcs in it, and a BSPT, which can similarly have at most n arcs. So the

FSPTs and BSPTs contribute in total no more than $2n$ arcs per breakpoint, giving a total of $2nK^{\text{nodes}}$ arcs. Adding to the waiting arcs, this gives a total of no more than $4nK^{\text{nodes}}$ arcs. Calculating the FSPT and BSPT for each breakpoint requires $\mathcal{O}(2 \times K^{\text{nodes}} \times SSP(n, |A|))$ operations, and, since the TENL is acyclic, (as all arcs point forwards in time), finding the shortest path in the TENL requires $\mathcal{O}(ASPP(2K^{\text{nodes}}n, 4K^{\text{nodes}}n))$ operations. The result follows. □

2.4.2 A DDD Algorithm for the MTTP

As for the MDP, we develop a dynamic discretization discovery algorithm to reduce the number of breakpoints explored whilst still being able to certify optimality.

Based on Proposition 1, ABSPTs are not the natural structures to use in an algorithm for the MTTP. An optimal solution consists of a sequence of travel subpaths connected by waiting, where each travel subpath can safely (by Proposition 1) be assumed to include a breakpoint and the waiting time between any two consecutive travel subpaths can safely be assumed positive (not zero). (By a "safe" assumption, we mean one that is satisfied by at least one optimal solution.) Under the former assumption, it must also be the case that each travel subpath consists of a TDSP to/from the breakpoint it includes. In the remainder of this chapter, we make this assumption, using the following definition.

Definition 10. *A* travel subpath for breakpoint (i, t)*, referred to as an* (i, t)*-travel subpath, is a waiting-free timed path concatenating a SP ending at* (i, t) *with a SP starting from* (i, t)*. The term* travel subpath *is used to refer to such a path for some unspecified breakpoint, while* i-travel subpath *is used for a travel subpath for a breakpoint at node* i *at an unspecified time.*

Thus there exists some optimal solution to an MTTP that has each of its travel subpaths lying in a TEN formed by the union of a forward and backward shortest path tree for a specific breakpoint. Hence, instead of an ABSPT, the more natural structure to use for solving the MTTP is the union of an i-FSPT and an i-BSPT for the same breakpoint at

node i. We call such a union a *mangrove*, and define it formally as follows. Note that we will use the union operator, $\cup$, to form a TEN as a union of two TENs, in the obvious way.

Definition 11. *A* mangrove *for breakpoint (i,t), denoted by $\mathcal{M}^{i,t}$, is the TEN created by taking the union of the i-FSPT $\mathcal{F}^{i,t}$ and the i-BSPT $\mathcal{B}^{i,t}$, written as $\mathcal{M}^{i,t} = \mathcal{F}^{i,t} \cup \mathcal{B}^{i,t}$. We refer to any such mangrove as an i-mangrove and use the prefix (i,t)- if we wish to specify the mangrove for breakpoint (i,t).*

Similarly to the BSPTs used in our MDP algorithm, the FIFO property ensures that for any node i, the set of all i-mangroves is totally ordered with respect to time for timed nodes in their i-FSPTs and for timed nodes in their i-BSPTs. Specifically, if $\mathcal{M}^{i,t} = \mathcal{F}^{i,t} \cup \mathcal{B}^{i,t}$ and $\mathcal{M}^{i,t'} = \mathcal{F}^{i,t'} \cup \mathcal{B}^{i,t'}$ for $t' > t$, then for any node j with (j,s) in $\mathcal{F}^{i,t}$ and (j,s') in $\mathcal{F}^{i,t'}$ it must be that $s' > s$, and, similarly, for any node j with (j,s) in $\mathcal{B}^{i,t}$ and (j,s') in $\mathcal{B}^{i,t'}$ it must be that $s' > s$.

As a consequence, any i-travel subpath that arrives at node i at time t or later has a representative timed node sequence in $\mathcal{M}^{i,t}$. However, to ensure its arcs are also represented, arc-completion is, again, needed. For given breakpoint (i,t), the *Arc-completed Forwards Shortest Path Tree* (AFSPT), denoted by $\overline{\mathcal{F}}^{i,t}$, is given by $\mathcal{F}^{i,t}$ together with the addition of arcs $((j,t'),(k,t''))$ that are not already in $\mathcal{F}^{i,t}$, for every $(j,k) \in A$ having both (j,t') and (k,t'') in the FSPT. Similarly, we extend the ABSPT definition: the ABSPT for breakpoint (i,t), denoted by $\overline{\mathcal{B}}^{i,t}$, is given by $\mathcal{B}^{i,t}$ together with the addition of arcs $((j,t'),(k,t''))$ that are not already in $\mathcal{B}^{i,t}$, for every $(j,k) \in A$ having both (j,t') and (k,t'') in the BSPT. The *Arc-completed Mangrove* for breakpoint (i,t) is denoted by $\overline{\mathcal{M}}^{i,t}$ and is the TEN formed by taking the union of $\overline{\mathcal{F}}^{i,t}$ and $\overline{\mathcal{B}}^{i,t}$. Note that to form an arc-completed mangrove, $\overline{\mathcal{M}}^{i,t}$, its FSPT and its BSPT are arc-completed separately, since we need only that the arc-completed mangrove represent travel subpaths containing node i at time t or later.

Our algorithm maintains a set of TENs, each either the mangrove or the arc-completed mangrove for some breakpoint. Each subset of these TENs that is induced by breakpoints at the same node are ordered by their breakpoint time. The algorithm thus maintains, for

each node i, an ordered list, denoted by L^i, of breakpoints, (i,t). Whenever (i,t) is added to L^i, the (i,t)-mangrove, $\mathcal{M}^{i,t}$, and its arc-completion, $\overline{\mathcal{M}}^{i,t}$, are computed. For each consecutive pair of breakpoints in L^i, say (i,t) and (i,t^+) are such a pair, with $t^+ > t$, we proceed as follows. In the case that there is no breakpoint at i between t and t^+, we say that the mangrove for (i,t) is *resolved*, or simply that (i,t) is resolved. In this case, by the same properties used for the MDP, it must be that if an i-travel subpath using a breakpoint in $[t,t^+)$ appears in the optimal solution, then it must lie in the (i,t)-mangrove. Thus the algorithm keeps only the (i,t)-mangrove, discarding any arcs in $\overline{\mathcal{M}}^{i,t} \setminus \mathcal{M}^{i,t}$, and sets the UTT to the true travel time, so

$$\underline{c}_{(j,s),(k,s')} := c_{j,k}(s)$$

for each timed arc $((j,s),(k,s'))$ in $\mathcal{M}^{i,t}$. Otherwise, in the case that (i,t) is *not* resolved, the algorithm sets the UTT on each arc $((j,s),(k,s'))$ in $\overline{\mathcal{M}}^{i,t}$ as follows: the UTT on each arc $((j,s),(k,s'))$ in $\overline{\mathcal{F}}^{i,t}$ (respectively $\overline{\mathcal{B}}^{i,t}$) is set to be

$$\underline{c}_{(j,s),(k,s')} := \min_{\tau} \{c_{j,k}(\tau) : s \leq \tau \leq s^+\}$$

where s^+ is the unique time such that (j,s^+) is a node in $\overline{\mathcal{F}}^{i,t^+}$ (respectively, $\overline{\mathcal{B}}^{i,t^+}$). Now any travel subpath for a breakpoint at i at a time in the interval $[t,t^+]$ is represented by a path in $\overline{\mathcal{M}}^{i,t}$ having total UTT no greater than the travel time of the travel subpath.

By ensuring that, for all i, L^i includes a sufficiently early breakpoint, any travel subpath is thus represented by a path in a mangrove or an arc-completed mangrove whose UTT is a lower bound on the travel subpath's travel time. For simplicity of exposition, our algorithm initializes L^i with $(i,0)$ for each i, but, in practice, it would be more efficient to use (i,t) for t the time of the first breakpoint at node i after s, where (i,s) is in $\mathcal{F}^{1,0}$. We similarly include (i,T) in the initial list, L^i, for each i, and deem the mangrove for (i,T) to be resolved, so the UTT for its arcs are simply their true travel time.

To obtain a lower bound on the travel time of the MTTP solution, which (without loss of generality) corresponds to a *sequence* of travel subpaths, the i-mangroves for different i must be connected, by arcs that represent waiting at a node between travel subpaths.

Our algorithm thus maintains a TEN, denoted by $\mathcal{D}_{LB}$, with associated UTT on each timed arc, defined as a function of the mangroves and arc-completed mangroves whose breakpoints are in the set $\bigcup_{i \in N} L^i$, as follows.

Timed Nodes and Travel Arcs. All timed nodes and all arcs in resolved mangroves and arc-completed mangroves are included in $\mathcal{D}_{LB}$, together with their UTT, as defined above.

Waiting Arcs. In addition, waiting arcs are included to connect the (arc-completed) mangrove[1] representing one travel subpath to another (arc-completed) mangrove that may represent the next. We consider how to represent any possible sequence consisting of an i-travel subpath ending at node j, then waiting at node j, followed by a k-travel subpath starting at node j, for $k \neq i$, while ensuring that the UTT of its representation is no greater than its true travel time. Recall that we need only consider the case of positive (nonzero) waiting time at j.

Specifically, for each timed node (j, s), with $j \neq n$, in the FSPT for some mangrove, say it is in $\mathcal{M}^{i,t}$, we consider representing a sequence starting with an i-travel subpath ending at j at time s or later, and hence visiting node i at time t or later. If $t = T$, then no (i, t)-travel subpath can appear in a feasible solution. So we assume $t < T$, and let s^+ be such that (j, s^+) is in the FSPT for the i-mangrove with the next breakpoint in L^i after (i, t). This must exist, since L^i is initialized to include (i, T). We consider two cases, where the second case has two subcases. These are illustrated in Figure 2.8 and Figure 2.9. In these figures, a square indicates a mangrove "root", a circle indicates a regular node, and travel subpaths in the network are shown as a wavy line.

Case 1. To represent waiting at j until time s^+ or later, for each node i, the timed arcs $((j, s), (j, s^+))$ are included in $\mathcal{D}_{LB}$, and assigned a UTT of zero. Waiting arcs join-

[1]Meaning a resolved mangrove or an arc-completed mangrove. Recall that mangroves and their arc-completions have the same timed node set.

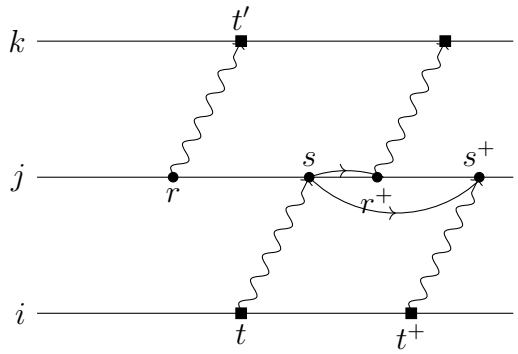


Figure 2.8: Forwards waiting arcs: Cases 1 and 2(a).

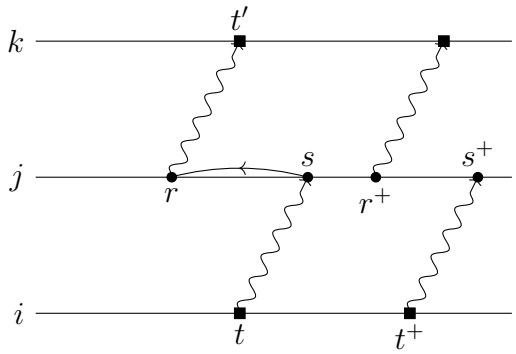


Figure 2.9: Backwards waiting arc: Case 2(b).

ing timed nodes in the BSPTs for consecutive i-mangroves in L^i are not needed, since we may, without loss of generality, assume that waiting occurs at the end of a travel subpath rather than at the start.

Case 2. For each node $k \neq i$, let r be the latest time that is earlier than s at which a timed node (j, r) appears in the BSPT for a k-mangrove, $\mathcal{M}^{k,t'}$, say. (So there is no $t'' > t'$ with $\mathcal{M}^{k,t''}$ in L^k, having (j, r') in $\mathcal{B}^{k,t''}$ with $r < r' < s$.) Such a t' and r must exist, since (j, s) in FSPT $\mathcal{F}^{i,t}$ implies that either $s > 0$ or $i = j$ and $s = t = 0$, while any timed node in BSPT $\mathcal{B}^{k,0}$ has negative time at any node other than k, and we initialize L^k to include $(k, 0)$. In addition, if $t' < T$, take r^+ so that (j, r^+) is in the BSPT for the mangrove with next breakpoint in L^k after (k, t'). Note that such $r^+ \geq s$. We consider two subcases.

(a) The mangrove for (k, t') is resolved. In this case, no waiting arc joining (j, s) to the mangrove for (k, t') is needed, since this mangrove represents only (k, t')-travel subpaths and the timed node (k, t') cannot be reached after positive waiting time at (j, s). If $t' = T$ then there is no next k-mangrove after that for (k, t'). Otherwise, (j, s) is connected to the next k-mangrove: the arc $((j, s), (j, r^+))$ is included in $\mathcal{D}_{LB}$, with a UTT of zero. This allows $\mathcal{D}_{LB}$ to represent the sequence of an i-travel subpath ending at j at time $\tau \in [s, s^+)$ followed by a

k-travel subpath starting at j at some time $\tau' > \tau$, using the next breakpoint at k after t'. Since $r^+ \geq s$ we call $((j,s),(j,r^+))$ a *forwards waiting arc*.

(b) The mangrove for (k,t') is not resolved. Then the arc $((j,s),(j,r))$ is included in $\mathcal{D}_{LB}$, with a UTT of zero. Then including the arc $((j,s),(j,r))$ allows $\mathcal{D}_{LB}$ to represent the sequence of an i-travel subpath ending at j at time $\tau \in [s, s^+)$ followed by a k-travel subpath starting at j at some time $\tau' \in [\tau, r^+)$. Since $r \leq s$ we call $((j,s),(j,r))$ a *backwards waiting arc*.

The way in which these waiting arcs allow the resulting TEN to represent possible optimal paths is illustrated in Figure 2.10, where possible optimal paths are drawn as a coiled line.

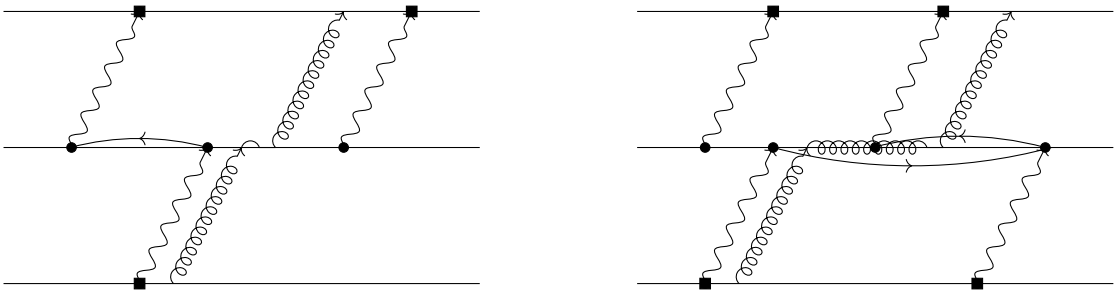


Figure 2.10: Lower bound paths using waiting arcs allow possible optimal paths to be represented.

Dummy Nodes and Arcs. Add a dummy start node and a dummy end node to $\mathcal{D}_{LB}$, and include in it arcs to connect the dummy start node to every timed node of the form $(1,t)$ with $t \geq 0$ and to connect every timed node of the form (n,t) with $t \leq T$ to the dummy end node, all arcs having zero UTT.

We claim that the least UTT path in $\mathcal{D}_{LB}$, constructed as above, from its dummy start to its dummy end node, has UTT a lower bound on the value of the MTTP. To see this, consider an optimal solution containing the sequence of an (i,α)-travel subpath, P_1^*, ending at (j,α') followed by a (k,β')-travel subpath, P_2^*, starting at (j,β), where $\beta - \alpha' > 0$ is the waiting time between the travel subpaths at node j. Let (i,s) be the latest breakpoint in L^i with $s \leq \alpha$ and let (k,t') be the latest breakpoint in L^k with $t' \leq \beta'$. Then P_1^* is represented

by a path in $\overline{\mathcal{M}}^{i,s}$ with UTT at most the travel time of P_1^* and P_2^* is represented by a path in $\overline{\mathcal{M}}^{k,t'}$ with UTT at most the travel time of P_2^*. Let (j, s') be the unique timed node in $\mathcal{F}^{i,s}$ for node j and let (j, t) be the unique timed node in $\mathcal{B}^{k,t'}$ for node j. We claim that there is a path in $\mathcal{D}_{LB}$ from (j, s') to (j, t) with UTT of zero, by the construction above. Observe that $s' \leq \alpha'$ and $t \leq \beta$ by the ordering of mangroves for the same node. However $t - s'$ may have any sign. If $t < s'$ then the (k, t')-mangrove cannot be resolved, (since otherwise it must be that $t' = \beta'$ and so $t = \beta > \alpha' \geq s'$), so the backward arc $((j, t), (j, s'))$ is included in $\mathcal{D}_{LB}$, as per Case 2(b). Otherwise, if $t \geq s'$, there are two subcases. In the case that the (k, t')-mangrove is resolved, the forward arc $((j, t), (j, s'))$ is included in $\mathcal{D}_{LB}$, as per Case 2(a). Otherwise, let (i, r) be the first breakpoint in L^i after (i, s) so that the unique timed node (j, r') in $\mathcal{F}^{i,r}$ has $r' \geq t$. (Such an r must exist since $j \neq n$ and (i, T) is in L^i.) Then by Case 1, there is a sequence of zero-UTT arcs connecting (j, s') to (j, r') and by Case 2(b) there is backwards arc $((j, r'), (j, s'))$ having zero UTT in $\mathcal{D}_{LB}$. Thus in all cases there is a zero-UTT path in $\mathcal{D}_{LB}$ from (j, s') to (j, t), and so the sequence P_1^* followed by P_2^* is represented in $\mathcal{D}_{LB}$ by a path having UTT no more than the combined travel time of P_1^* and P_2^*. We have thus proved the following result.

Proposition 3. *Using the procedure described above, let $\mathcal{D}_{\mathcal{LB}}$ and a UTT for each arc in $\mathcal{D}_{\mathcal{LB}}$ be constructed from lists L^i having the property that $(i, 0), (i, T) \in L^i$ for each $i \in N$. Then the least UTT path in $\mathcal{D}_{\mathcal{LB}}$ from its dummy start node to its dummy end node has UTT a lower bound on the value of the MTTP.*

In the statement of the algorithm, procedure $createLBTEN(\{L^i\}_{i \in N}, Resolved)$ constructs $\mathcal{D}_{LB}$ with its associated UTT, given in the vector $\underline{c}$, as described above, where $Resolved$ indicates the set of resolved breakpoints, and the procedure $computeSP(\mathcal{D}_{LB}, \underline{c})$ calculates the least UTT path, returning the path and its total UTT.

The lists $\{L^i\}_{i \in N}$ are also used to furnish an upper bound, by the construction of a TEN, denoted by $\mathcal{D}_{UB}$, as follows.

1. All arcs in (i,t)-mangroves for all (i,t) in L^i and all $i \in N$ are included in $\mathcal{D}_{UB}$, with their original travel times.

2. For each (j,s) in the FSPT for some i-mangrove, and for each $k \neq i$, let r be the earliest time later than s at which node (j,r) appears in the BSPT for a k-mangrove, if any. If r exists, add the waiting arc $((j,s),(j,r))$ to $\mathcal{D}_{UB}$, with travel time set to zero.

3. Add a dummy start node and a dummy end node to $\mathcal{D}_{UB}$, and include in it arcs to connect the dummy start node to every timed node of the form $(1,t)$ with $t \geq 0$ and to connect every timed node of the form (n,t) with $t \leq T$ to the dummy end node, all arcs having travel time set to zero.

Clearly any path in $\mathcal{D}_{UB}$ from its dummy start to its dummy end node corresponds to a feasible solution to the MTTP, and its total travel time gives an upper bound on the value of the MTTP. Minimizing the total travel time of such a path gives the best upper bound available from $\mathcal{D}_{UB}$. In the statement of the algorithm, procedure $createUBTEN(\{L^i\}_{i \in N})$ constructs $\mathcal{D}_{UB}$, and its associated travel times, stored in the vector $\overline{c}$, as described above, and the procedure $computeSP(\mathcal{D}_{UB}, \overline{c})$ calculates the current best feasible path for the MTTP, returning both the path and its total travel time.

The algorithm, stated formally as Algorithm 2, proceeds by solving a shortest path problem on both the upper bound and lower bound TEN to obtain upper and lower bounds on the value of the MTTP, respectively. If the two bounds are not equal, we consider the least UTT path in the lower bound TEN, and the arc-completed mangroves it uses. It must be that at least one of the arc-completed (i,t)-mangroves it uses has its breakpoint (i,t) not resolved, since otherwise the lower and upper bounds would be the same. The algorithm then chooses any set of one or more such breakpoints. For each breakpoint, (i,t) say, in the chosen set, some (one or more) breakpoints between (i,t) and the next breakpoint in L^i are selected, and added to L^i. If, now, any breakpoint in L^i is the first breakpoint at node i after

some breakpoint (i, s) in L^i, then (i, s) is marked as resolved. The lists $\{L_i\}_{i \in N}$ are again used to construct upper and lower bound TENs, and the bounds computed. This continues until the upper and lower bounds are equal. Alternatively, instead of computing an upper bound to decide whether the algorithm can terminate, the algorithm can terminate when all the arcs used in the lower-bound solution originate from resolved mangroves and provide a feasible and thus optimal solution to MTTP. This has computational advantages as only half as many shortest path calculations are performed. On the other hand, no duality gap is available during the execution of the algorithm. This variant is presented in Algorithm 3 and is the variant used in our computational experiments.

Clearly this algorithm is really a class of algorithms, and its performance in practice will depend on which set of unresolved breakpoints is chosen at each iteration, and, for each, which set of breakpoints at the same node are added to the list for that node. In Subsection 2.5.2 we describe and compare specific choices for these elements of the algorithm. Here, we simply name the function that decides a new set of breakpoints to add to the lists $FindBP(P, \{L^i\}_{i \in N})$, where P is the least UTT path found in $\mathcal{D}_{LB}$.

2.5 Computational Study

To analyze the performance of the algorithms presented in the previous section, we apply them to randomly generated instances. The algorithms are implemented in C++ and run on a Dell XPS Tower with Intel Core i9-9900 at 3.10GHz, 32GB RAM using a single thread. Our implementation of the algorithms as well as the instance data can be found at https://github.com/10heey1/TDSPP.

The shortest path procedures for calculating forward and backward shortest path trees are implemented using a Dijkstra-based algorithm similar to Orda and Rom (1990) with min-heap priority queue while the shortest path calculation on the overall time-expanded network uses a time-independent version of the same algorithm (since arc weights on the time-expanded network are time-independent). Our DDD algorithms are compared against

input : digraph $D = (N, A)$, latest time at end node T, arc travel time function $c_{i,j}(t)$ for all $t \in [0, T]$, for each $(i, j) \in A$
output: minimum travel time path from node 1 to n departing from 1 and arriving at n at times in $[0, T]$
$Resolved := \emptyset$;
for $i = 1$ **to** n **do**
  Compute the $(i, 0)$-mangrove and the arc-completed $(i, 0)$-mangrove;
  Compute the (i, T)-mangrove and the arc-completed (i, T)-mangrove;
  Set $L^i := ((i, 0), (i, T))$;
  $Resolved := Resolved \cup \{(i, T)\}$;
end
Initialize $LB = -\infty$ and $UB = \infty$;
while $(LB < UB)$ **do**
  Construct lower and upper bound TENs, with their UTT and travel times, respectively: $(\mathcal{D}_{LB}, \underline{c}) \leftarrow createLBTEN(\{L^i\}_{i \in N}, Resolved)$, $(\mathcal{D}_{UB}, \overline{c}) \leftarrow createUBTEN(\{L^i\}_{i \in N})$;
  Compute current lower and upper bounds along with their corresponding path: $(P_{LB}, LB) \leftarrow computeSP(\mathcal{D}_{LB}, \underline{c})$, $(P_{UB}, UB) \leftarrow computeSP(\mathcal{D}_{UB}, \overline{c})$;
  $BP \leftarrow findBP(P_{LB}, \{L^i\}_{i \in N})$;
  for $(j, t) \in BP$ **do**
    Compute the (j, t)-mangrove and the arc-completed (j, t)-mangrove;
    Insert (j, t) in the list L^j;
    if (j, t^-) *is in* L^j *immediately before* (j, t) *and no breakpoint at* j *is between them* **then**
      $Resolved := Resolved \cup \{(j, t^-)\}$
    end
    if (j, t^+) *is in* L^j *immediately after* (j, t) *and no breakpoint at* j *is between them* **then**
      $Resolved := Resolved \cup \{(j, t)\}$
    end
  end
end
return (UB, P_{UB})

Algorithm 2: *Dynamic Discretization Discovery* (DDD) Algorithm for the MTTP

their corresponding enumeration algorithm since no other exact polynomial time method exists for either the MDP and MTTP.

input : digraph $D = (N, A)$, latest time at end node T, arc travel time function $c_{i,j}(t)$ for all $t \in [0, T]$, for each $(i, j) \in A$
output: minimum travel time path from node 1 to n departing from 1 and arriving at n at times in $[0, T]$

$Resolved := \emptyset$;
for $i = 1$ **to** n **do**
  Compute the $(i, 0)$-mangrove and the arc-completed $(i, 0)$-mangrove;
  Compute the (i, T)-mangrove and the arc-completed (i, T)-mangrove;
  Set $L^i := ((i, 0), (i, T))$;
  $Resolved := Resolved \cup \{(i, T)\}$;
end
Initialize $LB = -\infty$;
while *true* **do**
  Construct the lower bound TEN, with their UTT:
  $(\mathcal{D}_{LB}, \underline{c}) \leftarrow createLBTEN(\{L^i\}_{i \in N}, Resolved)$;
  Compute the current lower bound along with their corresponding path:
  $(P_{LB}, LB) \leftarrow computeSP(\mathcal{D}_{LB}, \underline{c})$;
  $BP \leftarrow findBP(P_{LB}, \{L^i\}_{i \in N})$;
  if $BP = \emptyset$ **then**
    return (LB, P_{LB})
  end
  for $(j, t) \in BP$ **do**
    Compute the (j, t)-mangrove and the arc-completed (j, t)-mangrove;
    Insert (j, t) in the list L^j;
    if (j, t^-) *is in* L^j *immediately before* (j, t) *and no breakpoint at* j *is between them* **then**
      $Resolved := Resolved \cup \{(j, t^-)\}$
    end
    if (j, t^+) *is in* L^j *immediately after* (j, t) *and no breakpoint at* j *is between them* **then**
      $Resolved := Resolved \cup \{(j, t)\}$
    end
  end
end

Algorithm 3: *Dynamic Discretization Discovery* (DDD) Algorithm for the MTTP with no upper bound calculations

2.5.1 Instance generation

Network topology

To investigate the effect of network topology on the performance of the algorithms, we consider three different types of networks $D = (N, A)$ with node set $N = \{1, 2, \ldots, n\}$, and arc set described below. As before, let node 1 be the origin and node n be the destination. In addition, each arc is assigned a *base travel time* representing the free flow travel time of the arc, which is the travel time under the condition of no congestion. This will be used to create time-dependent travel time functions.

Type 1 (Corridor): Nodes are ordered from 1 to n. Arcs connect each node with up to three of the next eight nodes chosen randomly. Formally, for each $i \in N$, $(i, i+s) \in A$ and $(i+s, i) \in A$ for $s \in S_i$ where $S_i \subset \{1, 2, \ldots, 8\}$ and $|S_i| = 3$. If $i + s > n$, the arc $(i, i+s)$ and $(i+s, i)$ are not added (and no replacement arc is chosen). The base travel time of arc (i, j) is $|j-i| \times u \times c$ with $u = 0.5+U(0, 1)$, a uniform random number between 0.5 and 1.5, and c a constant proportional to Tn^{-1} and chosen such that travel times satisfy the FIFO property.

Type 2 (Grid): Nodes are organized into a rectangular grid, with the number of rows and number of columns in the grid differing by as little as possible, e.g., for a network with 500 nodes, the grid is chosen to be of dimension 25×20. The nodes are ordered first by row, then by column, so that a node i is to the left of node $i+1$ and above node $i + l$, where l is the length of the grid, provided that node i is not on the boundary of the grid. This ensures that the origin node 1 (top-left) and destination node n (bottom-right) are at opposite corners of the grid. The base travel time of an arc is $u \times c$ with $u = 0.5 + U(0, 1)$, a uniform random number between 0.5 and 1.5, and c a constant proportional to $T(l + h)^{-1}$ and chosen such that travel times satisfy the FIFO property. For this type of network, the number of arcs in the optimal solution is extremely likely to be $l + h$ where l is the length of the grid (the number of nodes

in each row *minus one*), and h is the height of the grid (the number of nodes in each column *minus one*).

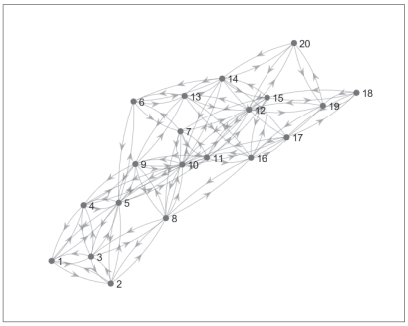
Type 3 (Triangle): Nodes are given by n (uniformly) random points in the unit square which are numbered in increasing order of their coordinate sums, so that the origin node 1 (bottom-left) and destination node n (top-right) are at opposite corners. The Delauney triangulation (dual graph of the Voronoi diagram) for the set of points is used to define the arcs of the network. The base travel time of arc i, j is $d_{ij} \times u \times c$ with d_{ij} the Euclidean distance between u and j, $u = 1 + U(0, 1)$, a uniform random number between 1 and 2, and c a constant proportional to T and chosen such that travel times satisfy the FIFO property.

Furthermore, for the all networks, arcs are bidirectional, so if (i, j) is an arc in the network (j, i) is also an arc in the network. Examples of the network types are shown in Figure 2.11, note that the figure only shows the network topology and thus arc lengths are not indicative of travel times. The three network types were chosen to resemble parts of street networks encountered in urban areas.

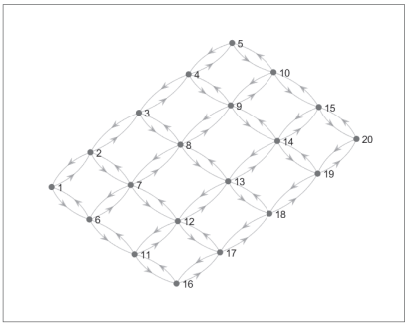
Time-dependent travel time functions

To investigate the effect of the travel time functions on the performance of the algorithms, we consider two types of travel time functions. The travel time functions are constructed, for each arc (i, j), to be the cubic spline, f, described below:

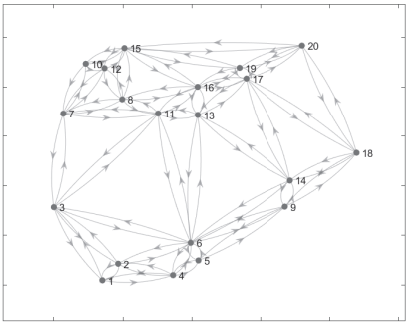
Type 1: f is a cubic spline, determined by fitting the values at the 9 points, given in



(a) Network Type 1 (Corridor)



(b) Network Type 2 (Grid)



(c) Network Type 3 (Triangle)

Figure 2.11: Examples of network types.

component-wise vector form, as

$$f(T_1) = \begin{cases} V_1 B_{i,j}, & \text{if } U_{i,j} < 1/3, \\ V_2 B_{i,j}, & \text{if } 1/3 \leq U_{i,j} < 2/3, \\ V_3 B_{i,j}, & \text{otherwise,} \end{cases}$$

where :

$T_1 = [0, T/8, T/4, 3T/8, T/2, 5T/8, 3T/4, 7T/8, T]$

$V_1 = [1.6, 1.3, 1, 1.025, 1.05, 1.025, 1, 1.3, 1.6]$

$V_2 = [2, 1.5, 1, 1.25, 1.5, 1.25, 1, 1.5, 2]$

$V_3 = [2.5, 1.75, 1, 1.375, 1.75, 1.375, 1, 1.75, 2.5]$

with $B_{i,j}$ is the base travel time of arc (i, j) and $U_{i,j} \sim U[0, 1]$.

Type 2: f is a cubic spline, determined by fitting the values at the 13 points, given in component-wise vector form, as

$$f(T_2) = \begin{cases} V_4 B_{i,j}, & \text{if } U_{i,j} < 1/3, \\ V_5 B_{i,j}, & \text{if } 1/3 \leq U_{i,j} < 2/3, \\ V_6 B_{i,j}, & \text{otherwise,} \end{cases}$$

where :

$T_2 = [0, T/8, T/4, 3T/8, T/2, 5T/8, 3T/4, 7T/8, T]$

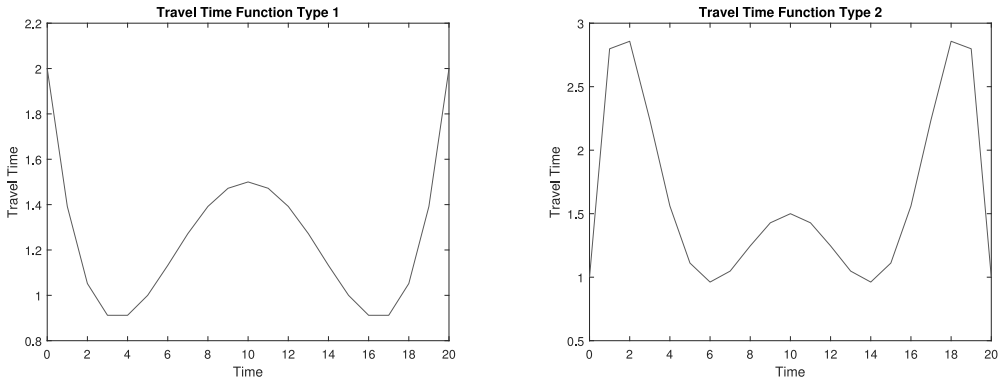
$V_4 = [1, 1.3, 1.6, 1.3, 1.025, 1.025, 1.05, 1.025, 1.025, 1.3, 1.6, 1.3, 1]$

$V_5 = [1, 1.5, 2, 1.5, 1.05, 1.25, 1.5, 1.25, 1.05, 1.5, 2, 1.5, 1]$

$V_6 = [1, 1.75, 2.5, 1.75, 1.05, 1.375, 1.75, 1.375, 1.05, 1.75, 2.5, 1.75, 1]$

with $B_{i,j}$ is the base travel time of arc (i, j) and $U_{i,j} \sim U[0, 1]$.

Hence, the parameter T is both an indicator of the length of the time horizon and the number of linear pieces of the travel time functions. An illustrative example of each of the two types is shown in Figure 2.12.



(a) Travel Time Type 1 (b) Travel Time Type 2

Figure 2.12: Plots of different travel time function types

The first type of travel time function is inspired by Figliozzi (2012), who constructs travel *speed* functions by dividing the time horizon into five intervals of equal length and assigning each of the five intervals a constant base time-independent travel speed. The base travel speed is taken and travel speeds approximately equal to 1.6, 1.0, 1.05, 1.0, and 1.6 times the base travel speed are assigned to the first, second, third, fourth, and fifth interval, respectively. In the same spirit, we have travel *time* functions that are approximately equal to 1.6, 1.0, 1.05, 1.0, and 1.6 times the base travel time at times 0, $1/4\ T$, $1/2\ T$, $3/4\ T$, and T, respectively. However, to create a continuous travel time function, we take the cubic spline through these points and additional points were added to control the shape of the function. It is then converted into a piecewise linear function by taking the piecewise linear interpolant of the spline sampled at integer points. The second type of travel time function is an extension of the first with two additional troughs for added difficulty. The two travel time functions were chosen to resemble the behavior in a road network in a ten-hour period containing both the morning and evening rush hours. We have experimented with other travel time functions proposed in Figliozzi (2012), but found that with these travel

time functions, the algorithms find both the minimum duration and minimum travel time solutions in a few iterations. In particular, we found that with these travel time functions the optimal solutions typically depart at the earliest or at the latest possible time. Thus these travel time functions are not very helpful for developing insights into the performance of the algorithms.

The performance of both DDD algorithms critically depends on the calculation of UTT: being able to efficiently compute the minimum arc travel time in a start time interval. This can be accomplished by efficiently pre-computing a look-up table which contains only the time points which are local minima of the travel time.

2.5.2 Algorithm configuration

In this section, we investigate the impact of the choice of the next breakpoint to explore on the performance of the algorithms. For the MDP, suppose the current lower bound is given by LB^t and t^+ is the arrival time at n of the next ABSPT in the list after $\overline{\mathcal{B}}^t$. Denote timed nodes in $\overline{\mathcal{B}}^t$ by (i, t_i) and timed nodes in $\overline{\mathcal{B}}^{t^+}$ by (i, t_i^+). We select (i, j) to be an outgoing arc from a timed node in the ABSPT $\overline{\mathcal{B}}^t$ that has a breakpoint in its travel time function within the interval (t_i, t_i^+), and that is first encountered when checking the nodes from 1 to n, in index order. We consider the following three alternative schemes for selecting one of the breakpoints of $c_{i,j}(t)$ lying within (t_i, t_i^+):

1. pick a random breakpoint (RAND),
2. pick the median breakpoint (with the breakpoints chronologically listed) (MED), and
3. pick the minimizer of $c_{i,j}(t)$ over t a breakpoint with $t \in (t_i, t_i^+)$ (MIN).

For the MTTP, we furthermore compare adding a single breakpoint (S), which is a breakpoint for one of the travel subpaths, in each iteration, with adding multiple breakpoints (M), by adding a breakpoint for each of the travel subpaths, in each iteration. To select which breakpoint(s) to add, we identify the sequence of arc-completed mangroves used by

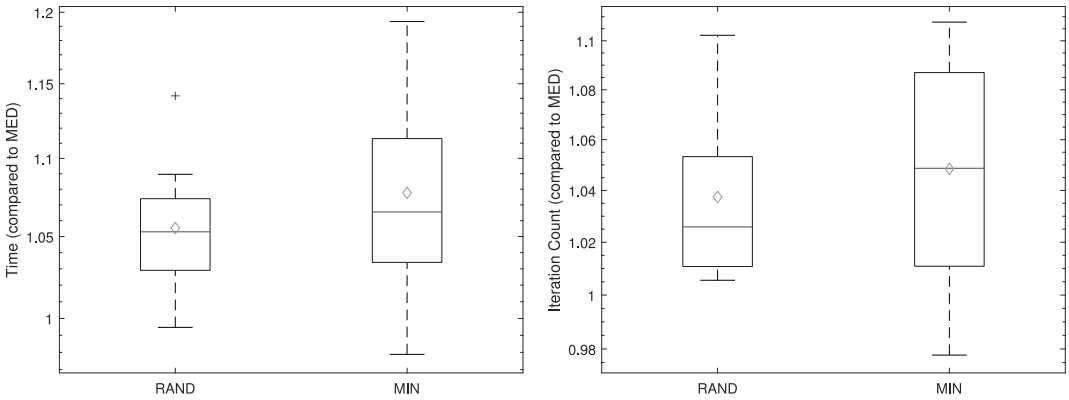
the UTT path, P_{LB}. (Each travel subpath of P_{LB} occurs in an arc-completed mangrove.) For such an arc-completed mangrove, let (i, t) be the breakpoint used to generate it. Then, if (i, t^+) is the next element in L^i after (i, t), and there is a breakpoint (not yet explored) at i between t and t^+, we may select one such breakpoint to explore; such breakpoints are candidates for selection. Which one is selected is dictated by the scheme used: RAND, MED or MIN. In the case of the (S) strategy, a breakpoint at i is selected for only the first arc-completed i-mangrove used by the UTT path, encountered in traversing it from node 1 to node n, for which a candidate breakpoint exists. In the case of the (M) strategy, a breakpoint is added for every arc-completed mangrove used by the UTT path for which a candidate breakpoint exists.

Computational experiments show that when solving instances of the MTTP almost all computing time ($> 90\%$) is spent solving (time-independent) shortest path problems on the TEN used for lower bound computations. Although each instance of the shortest path problem is smaller than the instances solved by the enumeration algorithm (as evidenced by the number of breakpoints explored), the larger number of shortest path solves negates the gains per solve. Therefore, for solving MTTP, we introduce a strategy that adds breakpoints more aggressively in order to reduce the number of shortest path solves. More specifically, the aggressive $findBP$ procedure (AGG) adds for each (i, t_i)-mangrove not only the (i, t_i^{MIN})-mangrove for the minimum arc travel time but also for the next breakpoint $(i, t_i + 1)$, the last breakpoint in the interval $(i, t_i^+ - 1)$, the breakpoints surrounding (i, t_i^{MIN}), i.e., $(i, t_i^{MIN} + 1)$ and $(i, t_i^{MIN} - 1)$, and the breakpoint and surrounding breakpoints for other local minima. For the travel time functions used in our computational experiments, this means adding up to 8 breakpoints per mangrove used in the travel subpaths of the lower bound solution.

First, we consider the impact of the breakpoint selection scheme when solving MDP instances. The results can be found in Figure 2.13 where we show box-plots of the ratios solve time, iteration count, and number of breakpoints explored for schemes RAND and

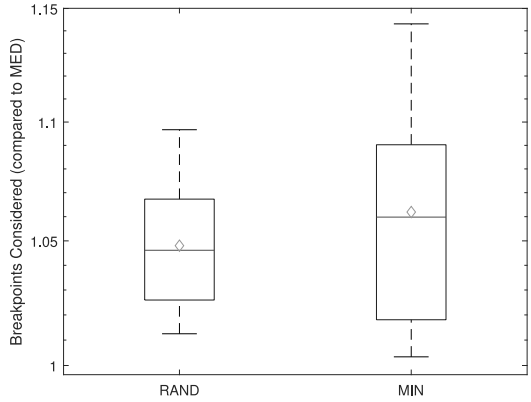
MIN compared to scheme MED.

The mean of each box-plot is represented by a diamond and the median by a horizontal line. We observe that the best choice among the three breakpoint selection scheme is MED, which does best on average in all performance metrics (solve time, number of iterations, and number of breakpoints explored). Thus we choose scheme MED for our computational experiments.



(a) Effect on solve time

(b) Effect on iteration count

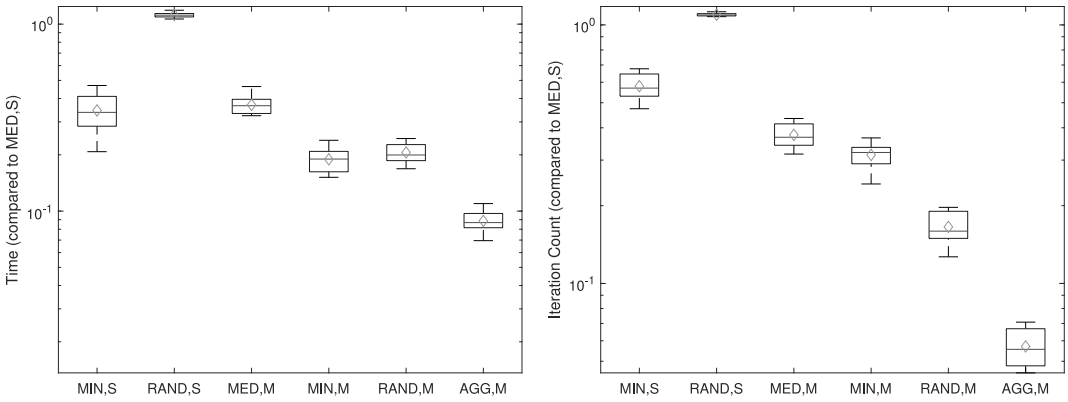


(c) Effect on number of breakpoints explored

Figure 2.13: Comparison of breakpoint selection schemes on 120 instances of the MDP with $n = 50$, $T = 40, 60$

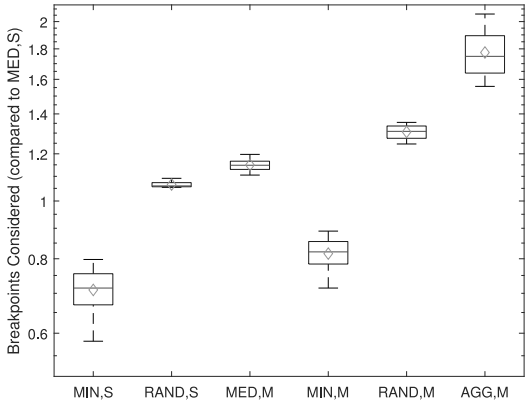
Next, we consider the impact of the breakpoint selection scheme and the impact of the number of breakpoints explored each iteration when solving MTTP instances. The results can be found in Figure 2.14 where we show box-plots of the ratios of solve time, iteration count, and number of breakpoints explored for schemes (RAND, S), (RAND, M), (MED,

M), (MIN, S), (MIN, M), and (AGG, M) compared to scheme (MED, S). (Note that the values on the y-axis are given in a logarithmic scale.)



(a) Effect on solve time

(b) Effect on iteration count



(c) Effect on number of breakpoints explored

Figure 2.14: Comparison of breakpoint selection schemes on 120 instances of the MTTP with $n = 50$, $T = 40, 60$

We observe that scheme (MIN, S) results in the minimum number of breakpoints explored, but that scheme (AGG, M) results in the minimum number of iterations and solve time; there is a trade-off between iteration count and number of breakpoints considered and these two metrics affect the solve time multiplicatively. We choose scheme (AGG, M) to use in the following computational experiments.

2.5.3 Benefits of dynamic discretization discovery

In this section, we compare the performance of our *Dynamic Discretization Discovery* (DDD) algorithms to the performance of the breakpoint enumeration algorithms, where we use breakpoint selection scheme MIN and, in the case of MTTP, we explore multiple breakpoints in each iteration.

The results for MDP and MTTP can be found in Table 2.1 and Table 2.2 respectively, where we present averages over 10 randomly generated instances for $n = 50$, $T \in \{20, 40, 60, 80, 100\}$, the three types of networks, and the two types of travel time functions. The values of T were chosen to reflect 30 minute ($T = 20$), 15 minute ($T = 40$) and finer ($T = 60$, 80, 100) discretizations if the travel time functions are contained in a 10 hour interval. Even though it appears we are increasing the length of the time horizon, since the travel time functions are scaled accordingly (there is a factor of T), this is equivalent to increasing the discretization without increasing the time horizon. For example, if the time horizon doubles in length, then so does the travel time, the only difference is that there are now twice as many discretization points. We report the number of breakpoints explored by the DDD algorithm (BP), the number of breakpoints explored by the enumeration algorithm (i.e., the number of breakpoints in the instances, Total # BP), and the fraction of the number of breakpoints investigated by the DDD algorithm (given as a percentage, % BP). Furthermore, we report the solve time in milliseconds for both the DDD algorithm (Time DDD (ms)), the enumeration algorithm (Time Enum (ms)) and the solve time of the DDD as a fraction of the solve time of the enumeration algorithm (given as a percentage, % Time), and the number of arcs in the optimal solution (# Arcs). Note that for these instances the number of breakpoints is $|N - 1| \times (T - 1) + 2$, and not $|A| \times (T - 1) + 2$ as in Foschini et al. (2014), since for arcs with a common tail node and breakpoint, we only need to explore the breakpoint once. In the case of MTTP, we also report the number of subpaths in the optimal solution (# Subpaths) since this is a critical factor in the difficulty of MTTP.

Table 2.1: MDP results

T	ntype	ttype	BP	Total # BP	% BP	Time DDD (ms)	Time Enum (ms)	% Time	# Arcs
20	C	1	37.4	933	4.0	13.3	197.0	6.8	7.5
40	C	1	49.0	1913	2.6	17.6	389.4	4.5	7.6
60	C	1	57.3	2893	2.0	20.6	587.4	3.5	7.6
80	C	1	68.4	3873	1.8	25.0	789.0	3.2	7.6
100	C	1	73.7	4853	1.5	28.3	1010.6	2.8	7.6
20	C	2	51.6	933	5.5	18.4	214.7	8.6	7.5
40	C	2	54.9	1913	2.9	19.1	423.3	4.5	7.5
60	C	2	64.9	2893	2.2	23.9	636.7	3.8	7.5
80	C	2	71.1	3873	1.8	25.8	860.9	3.0	7.5
100	C	2	83.0	4853	1.7	31.7	1079.0	2.9	7.5
20	G	1	48.6	933	5.2	8.2	110.2	7.4	13.0
40	G	1	53.9	1913	2.8	9.2	217.9	4.2	13.0
60	G	1	71.4	2893	2.5	12.4	331.6	3.7	13.0
80	G	1	80.8	3873	2.1	14.0	444.9	3.1	13.0
100	G	1	91.0	4853	1.9	16.0	558.3	2.9	13.0
20	G	2	47.6	933	5.1	8.3	119.1	7.0	13.0
40	G	2	58.6	1913	3.1	9.8	238.0	4.1	13.0
60	G	2	71.7	2893	2.5	12.4	362.3	3.4	13.0
80	G	2	82.2	3873	2.1	14.1	487.2	2.9	13.0
100	G	2	94.9	4853	2.0	16.6	609.9	2.7	13.0
20	T	1	47.9	933	5.1	15.2	162.7	9.3	9.0
40	T	1	59.1	1913	3.1	18.1	321.1	5.6	9.0
60	T	1	69.7	2893	2.4	21.0	488.3	4.3	9.0
80	T	1	81.5	3873	2.1	25.0	653.8	3.8	9.0
100	T	1	94.8	4853	2.0	28.2	822.7	3.4	9.0
20	T	2	42.8	933	4.6	12.6	173.8	7.2	8.9
40	T	2	53.7	1913	2.8	16.4	349.0	4.7	8.9
60	T	2	66.3	2893	2.3	20.2	531.6	3.8	8.9
80	T	2	81.9	3873	2.1	26.2	722.7	3.6	8.9
100	T	2	87.8	4853	1.8	26.6	902.2	2.9	8.9

Table 2.2: MTTP results.

T	ntype	ttype	BP	Total # BP	% BP	Time DDD (ms)	Time Enum (ms)	% Time	Optimal Path # Arcs	Optimal Path # Subpaths
20	C	1	258.2	933	27.7	850.7	562.0	151.4	7.6	3.9
40	C	1	415.7	1913	21.7	1240.9	1240.8	100.0	7.6	4.1
60	C	1	581.8	2893	20.1	1782.6	2154.2	82.8	7.6	3.8
80	C	1	750.0	3873	19.4	2433.1	3218.9	75.6	7.6	3.6
100	C	1	916.8	4853	18.9	3092.9	4398.8	70.3	7.6	4.1
20	C	2	257.2	933	27.6	728.7	562.4	129.6	7.5	4.9
40	C	2	421.7	1913	22.0	1146.3	1325.9	86.5	7.5	5.5
60	C	2	592.5	2893	20.5	1633.6	2325.8	70.2	7.5	5.2
80	C	2	733.9	3873	18.9	1979.0	3491.6	56.7	7.5	5.2
100	C	2	897.7	4853	18.5	2512.4	4724.3	53.2	7.5	5.3
20	G	1	364.4	933	39.1	1793.1	494.1	362.9	13.0	4.6
40	G	1	581.6	1913	30.4	2506.2	1228.1	204.1	13.0	4.6
60	G	1	905.9	2893	31.3	4657.2	2198.9	211.8	13.0	4.1
80	G	1	1185.9	3873	30.6	6625.8	3348.4	197.9	13.0	3.7
100	G	1	1364.5	4853	28.1	7253.0	4623.4	156.9	13.0	4.0
20	G	2	406.8	933	43.6	2141.4	534.3	400.8	13.0	5.9
40	G	2	582.3	1913	30.4	2149.0	1318.4	163.0	13.0	6.7
60	G	2	809.5	2893	28.0	3014.4	2377.8	126.8	13.0	6.4
80	G	2	1055.0	3873	27.2	4078.2	3617.7	112.7	13.0	6.2
100	G	2	1301.0	4853	26.8	5239.4	4988.0	105.0	13.0	6.5
20	T	1	277.2	933	29.7	1060.3	476.9	222.3	8.7	4.6
40	T	1	430.7	1913	22.5	1351.2	1164.4	116.0	8.7	4.4
60	T	1	619.8	2893	21.4	2124.4	2068.7	102.7	8.7	3.8
80	T	1	800.3	3873	20.7	2878.3	3123.8	92.1	8.7	3.8
100	T	1	982.0	4853	20.2	3686.9	4267.5	86.4	8.7	4.2
20	T	2	304.0	933	32.6	1226.8	509.6	240.7	8.6	4.9
40	T	2	456.6	1913	23.9	1362.5	1255.5	108.5	8.7	5.1
60	T	2	621.8	2893	21.5	1841.9	2216.4	83.1	8.7	5.1
80	T	2	801.9	3873	20.7	2474.0	3344.6	74.0	8.7	5.0
100	T	2	991.2	4853	20.4	3216.1	4550.3	70.7	8.7	5.4

The results can be found in Table 2.1 and Table 2.2 and show that the DDD algorithm investigates only a small fraction of the number of breakpoints for the MDP (between 1.5% and 5.5%) and a larger fraction for the MTTP (18.5% to 43.6%) with, as expected, the fraction decreasing when the fineness of the discretization (T) increases. For the MDP this translates into significantly smaller computing times (compared to the enumeration algorithm). For the MTTP the benefits, in terms of computing time, are smaller and depend on the class of instances. For instances with a corridor and triangle topology and for finer time discretizations the DDD algorithm outperforms the enumeration algorithm, but for instances with a grid topology the enumeration algorithm outperforms the DDD algorithm. While the results for the MTTP are mixed, it is clear that the DDD algorithm becomes more competitive as T increases. Furthermore, our implementation does not take advantage of the fact that the TEN does not change much between consecutive iterations and always solves shortest path problems from scratch. A tailored shortest path algorithm, which keeps track of labels and, possibly, the paths that gave rise to those labels, may provide significant computational advantages. (With the use of a tailored shortest path algorithm, schemes such as (MIN,S) which explore fewer breakpoints may yield the best results.)

To see how well the DDD algorithm for the MDP scales, we have conducted a final computational experiment in which we randomly generated 5 (large) instances for each combination of network type and travel time function for networks of size $n = 1000$ and $n = 10000$.

The results are shown in Table 2.3 (where we report the same statistics as in Table 2.1 except that units of time are now in seconds instead of milliseconds). We do not report results for the enumeration algorithm for the instances with $n = 10000$ as there was insufficient memory to accommodate these instances.

We see that the DDD algorithm scales well. It investigates only a tiny fraction (0.1% to 3.4%) of the number of breakpoints, and requires much less computing time than the enumeration algorithm (0.4% to 5.7%). The results demonstrate that the fraction of explored

Table 2.3: MDP results for large n.

n	T	ntype	ttype	BP	Total # BP	% BP	Time DDD (s)	Time Enum (s)	% Time	# Arcs
1000	20	C	1	403.4	18983	2.1	4.9	130.0	3.8	150.2
1000	40	C	1	406.0	38963	1.0	4.4	233.9	1.9	150.0
1000	60	C	1	372.0	58943	0.6	4.0	330.6	1.2	150.0
1000	80	C	1	365.8	78923	0.5	3.6	475.3	0.8	150.0
1000	100	C	1	380.0	98903	0.4	4.4	742.2	0.6	150.0
1000	20	C	2	647.6	18983	3.4	7.8	136.0	5.7	150.2
1000	40	C	2	414.8	38963	1.1	4.1	232.1	1.8	150.2
1000	60	C	2	359.0	58943	0.6	3.2	330.7	1.0	150.2
1000	80	C	2	360.4	78923	0.5	4.1	598.2	0.7	150.2
1000	100	C	2	358.0	98903	0.4	4.6	831.3	0.6	150.4
1000	20	G	1	202.0	18983	1.1	1.3	77.0	1.7	63.0
1000	40	G	1	235.8	38963	0.6	1.4	148.1	1.0	63.0
1000	60	G	1	279.0	58943	0.5	1.5	199.9	0.8	63.0
1000	80	G	1	301.4	78923	0.4	1.9	353.0	0.5	63.0
1000	100	G	1	325.6	98903	0.3	2.2	482.7	0.5	63.0
1000	20	G	2	190.2	18983	1.0	1.3	86.2	1.5	63.0
1000	40	G	2	222.8	38963	0.6	1.3	163.6	0.8	63.0
1000	60	G	2	257.4	58943	0.4	1.4	223.5	0.6	63.0
1000	80	G	2	299.0	78923	0.4	2.1	474.9	0.4	63.0
1000	100	G	2	326.0	98903	0.3	2.0	536.1	0.4	63.0
1000	20	T	1	249.6	18983	1.3	2.4	106.3	2.3	45.0
1000	40	T	1	233.0	38963	0.6	2.3	220.6	1.0	45.0
1000	60	T	1	246.2	58943	0.4	2.2	310.4	0.7	45.0
1000	80	T	1	280.0	78923	0.4	2.9	525.8	0.6	45.0
1000	100	T	1	326.8	98903	0.3	3.5	664.7	0.5	45.0
1000	20	T	2	226.4	18983	1.2	2.2	120.2	1.8	43.6
1000	40	T	2	237.6	38963	0.6	2.3	240.1	1.0	43.8
1000	60	T	2	270.2	58943	0.5	2.4	339.3	0.7	43.8
1000	80	T	2	320.4	78923	0.4	3.7	608.2	0.6	43.8
1000	100	T	2	354.8	98903	0.4	4.7	835.9	0.6	43.8
10000	40	C	1	3423.0	389963	0.9	589.1	-	-	1481.8
10000	60	C	1	2923.6	589943	0.5	701.8	-	-	1482.2
10000	40	C	2	3519.4	389963	0.9	666.8	-	-	1484.2
10000	60	C	2	2735.6	589943	0.5	814.2	-	-	1484.4
10000	40	G	1	684.6	389963	0.2	102.1	-	-	198.0
10000	60	G	1	872.0	589943	0.1	187.2	-	-	198.0
10000	40	G	2	754.0	389963	0.2	125.0	-	-	198.0
10000	60	G	2	891.0	589943	0.2	178.1	-	-	198.0
10000	40	T	1	761.0	389963	0.2	186.0	-	-	137.4
10000	60	T	1	901.4	589943	0.2	264.7	-	-	137.0
10000	40	T	2	810.0	389963	0.2	200.6	-	-	135.4
10000	60	T	2	833.6	589943	0.1	242.0	-	-	135.4

breakpoints decreases as a function of the number of nodes in the network. This reflects the benefit of using the ABSPT, which is able to represent all node breakpoints simultaneously. Not surprisingly the network topology appears to have a large impact on the performance of the DDD algorithm, with the corridor instance being the most challenging, most likely due to the number of arcs in the optimal solution. The results also clearly show that the algorithm becomes more competitive as the discretization becomes finer. Finally, note that our results show that the use of a finer discretization does not necessarily produce instances that are harder to solve for the DDD algorithm. For example, for the Corridor instances we see that the solution time for instances with $T = 60$ is smaller than the solution time for instances with $T = 40$. This may be due, in part, to the use of the breakpoint selection scheme, which appears to "identify" the optimal solution earlier when a finer discretization is used.

2.6 Real-world case study

To supplement the study of random instances, we also run the DDD algorithm for the MDP on real-world data from part of the Atlanta road network, which comprises 306 nodes and 618 arcs and is obtained via Open Street Map. A map view of the area as well as the corresponding network representation are shown in Figure 2.15.

The arc travel times were derived from GPS traces obtained from a variety of input streams (e.g., navigational systems and cell phones) and snapped to the nearest arc. The collected data are then aggregated to create piecewise linear arc travel time functions with breakpoints at every 15 minutes for a 24-hour period, giving a total of 192 data points for each arc, similarly to how travel time functions are created in Marshall (2019). For a sample of the arc travel time functions, see Figure 2.16.

To illustrate the difference between a minimum duration, an earliest arrival, and a latest departure paths from an origin to a destination in a specific time interval, we selected an origin in North Atlanta, a destination in Atlanta's Midtown, and a time interval from 9:00am

(a) Map of section of Midtown and North Atlanta

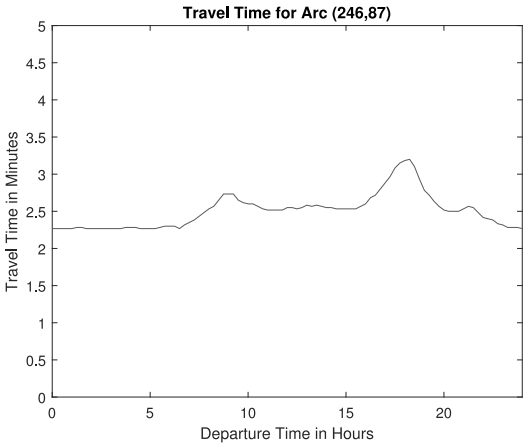
(b) Network representation

Figure 2.15: Map and corresponding network representation of the section of Midtown and North Atlanta.

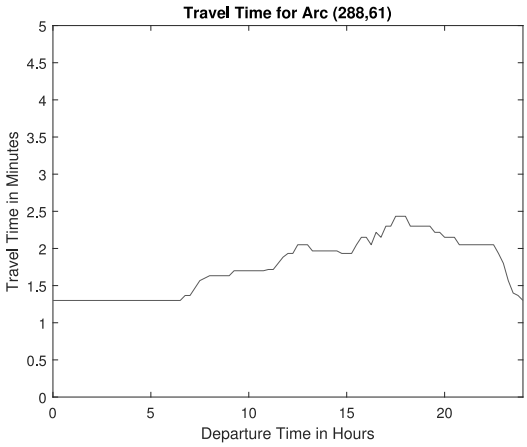
to 1:00pm. The earliest arrival path starts at the origin at 9:00am, the latest arrival path ends at the destination at 1:00pm, and the minimum duration path can start at any time within the [09:00, 13:00] interval.

The minimum duration path (see Figure 2.17b) leaves at 10:37am and takes 45.8 minutes. It is different from both the earliest arrival and latest departure paths, which coincide (see Figure 2.17a), and take 47.5 and 47.8 minutes, respectively. The minimum duration path departs in the lull between the morning and noon rush hours, and uses the most direct path (taking the interstates I-75 and I-75/85), whereas the earliest arrival/latest departure path avoids the interstates and instead uses streets in residential neighborhoods in the eastern part of Midtown (Peachtree St. and Juniper St.). The apparent detour at the end of the path is due to one-way streets.

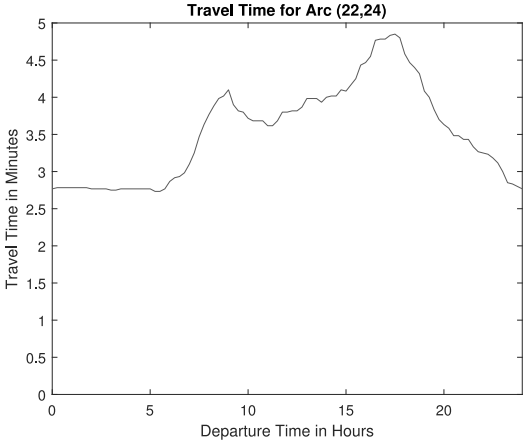
Although the above example illustrates that the minimum duration path can be different from the earliest arrival and latest departure paths, we have found that the difference in duration is not that large. This is likely due to the fact that the chosen section of Atlanta is relatively small. Furthermore, this section has few schools, and school zones are known to



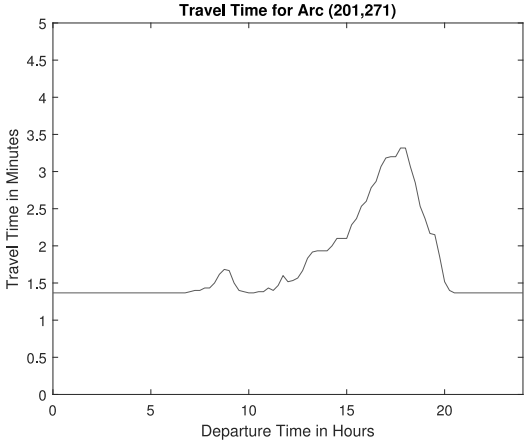
(a) Travel time of an arc along Peachtree St.



(b) Travel time of an arc along Ponce de Leon Ave. NE



(c) Travel time of an arc along North Ave. NE



(d) Travel time of an arc along Midtown section of I-75/85

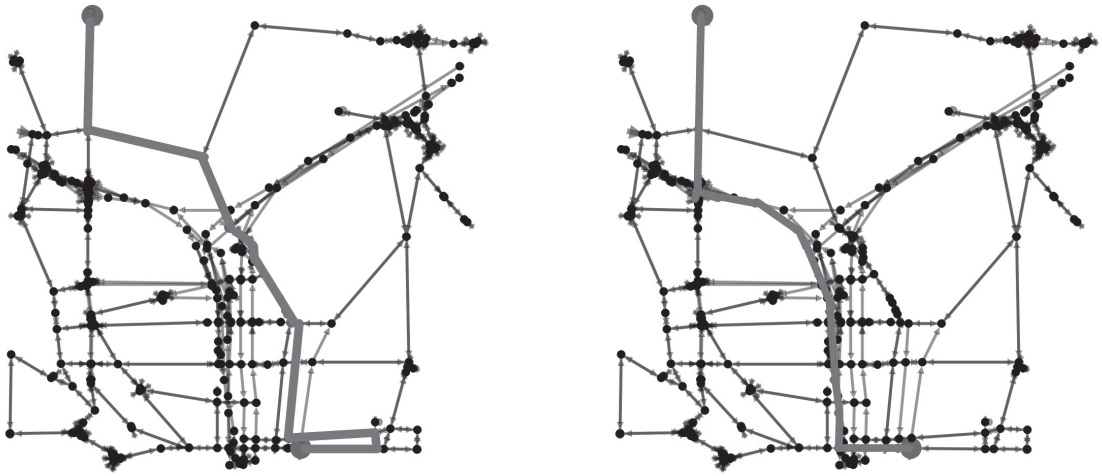
Figure 2.16: Travel times for four different arcs in Atlanta.

greatly influence travel times at certain times of the day.

Finding the minimum duration path in the network representing this section of Atlanta took a few seconds, which shows that the technology developed might be useful in real-life applications.

2.7 Concluding Remarks

We have generated new dynamic discrete discovery algorithms for time-dependent shortest path problems where the travel time on an arc is a piecewise linear function of the departure



(a) Earliest arrival / latest departure path

(b) Minimum duration path

Figure 2.17: An example where the minimum duration path differs from the earliest arrival/latest departure path.

time on the arc. These algorithms make use of the breakpoints of the piecewise linear functions and the previous result of the existence of an algorithm polynomial in the total number of breakpoints for the minimum duration objective. However our algorithms, in contrast with the earlier result, investigate only a small fraction of the travel time function breakpoints in searching for an optimal path and proving its optimality.

The difference in computational benefits of the DDD algorithms for MDP and MTTP is not unexpected. For MTTP, not only is there no longer a natural decoupling, but there is also no unifying structure, like the ABSPT for the MDP, that can capture the breakpoints of each node. The larger fraction of breakpoints explored by the DDD algorithm for the MTTP is partially due to requiring $2n$ mangroves in the first iteration (compared to only 2 ABSPT for the MDP). For smaller values of T, this represents a sizeable fraction of the total number of breakpoints. That is not to say that the DDD algorithm for the MTTP is without merit, since it does reduce the number of breakpoints explored. Fine discretizations, and instances for which the optimal path consists of few subpaths and arcs, appear to be where the DDD algorithm has the greatest benefit. Another observation is that the enumeration algorithm for the MTTP is less affected by the network type, with similar solve times for all

network types and travel time function types. This is likely due to the fact that the number of nodes in the full time-expanded network is the same for the three network types. This is a consequence of the connectivity of the networks. It is possible to get from any i to any j in all network types, and, therefore, the mangroves are all the same size.

The results clearly show that developing an effective DDD algorithm for MTTP is significantly more challenging than developing one for MDP and additional ideas (as well as improved implementations) are needed to be able to efficiently solve large instances. One opportunity may be to exploit the observation that in street networks it is not feasible to wait at certain nodes. It may be possible to model these nodes implicitly in the travel time functions. We leave the exploration of these and other ideas for future research.

CHAPTER 3
TIME-DEPENDENT SHORTEST PATH PROBLEMS WITH PENALTIES AND LIMITS ON WAITING

3.1 Introduction

Time-dependent shortest path problems generalize the classical problem of finding a path in a network from a given origin node to a given destination node so as to minimize a function of the arcs used, by introducing the element of time. A time-dependent path is a path in time and space (the network) that departs the origin at a time no earlier than the start of a given time horizon and reaches the destination no later than the end of the horizon, where the time to traverse an arc in the network is a function of the time of departure at its tail node. Such problems are especially of interest in road networks, where traffic congestion conditions affect the time needed to traverse links in the network (Letchner et al. 2006, Franceschetti et al. 2018).

Applications in transport logistics are emerging, in part because the now ubiquitous GPS-enabled devices supply the data needed to reliably estimate arc travel time functions in road networks (Bertsimas et al. 2019), complementing the more traditional reliance on data from inductive loops (Wang and Nihan 2000, Coifman 2002).

A number of variants of the time-dependent shortest path problem have been studied, starting with the *minimum arrival time* variant in Cooke and Halsey (1966), Orda and Rom (1990), and Dean (2004b), the *minimum duration* variant in Orda and Rom (1990), Chabini (1998), Nachtigall (1995), Kanoulas et al. (2006), Ding et al. (2008), Foschini et al. (2014) and He et al. (2019), and, recently, the *minimum travel time* variant in He et al. (2019). These variants differ not only in their choice of objective function, but also in the nodes at which waiting may occur in an optimal solution. In the *Minimum Travel Time Problem*

(MTTP), an optimal solution may wait at any node, including the origin, to take advantage of shorter travel times that occur later in the time horizon. The MTTP objective is to minimize the sum of the travel times along the arcs used; provided the path starts and ends within the given time horizon, time spent waiting is ignored. In the *Minimum Arrival Time Problem* (MATP), a path reaching the destination as soon as possible is sought. Under the commonly assumed *First-In First-Out* (FIFO) property of arc travel time functions, which states that waiting at the tail node of an arc can never result in an earlier arrival at its head node, waiting before departing a node is suboptimal for the MATP. In the *Minimum Duration Problem* (MDP) under FIFO, a path minimizing the difference between the arrival time at the destination and the departure time at the origin is sought. Thus, waiting to depart at any node other than the origin is suboptimal for the MDP. It has been shown that for piecewise linear travel time functions satisfying FIFO, all three variants can be solved in polynomial time (Cooke and Halsey 1966, Foschini et al. 2014, He et al. 2019). We note that early algorithms on variants in which waiting can be beneficial, e.g., the ones in Chabini (1998) and Dean (2004a), rely on time discretization, and, thus, provide only approximate solutions, whereas recent algorithms, e.g., the ones in Foschini et al. (2014) and He et al. (2019), however, properly handle continuous time and provide optimal solutions.

That the study of different objective functions is not only of academic interest can be seen in Cooper and Cowlagi (2018). They consider the problem of planning a route in which the objective is to minimize a "weighted sum of travel duration and exposure to traffic", and argue that such an objective may be of importance to reduce the health risks for long-haul truck drivers by reducing their exposure to emissions. They observe that because real-time traffic data is becoming widely available, which allows accurate predictions of traffic, and, thus, travel times, an optimal route may involve waiting for traffic to subside. They also comment that this type of objective may be relevant for motion-planning of aerial vehicles in inclement weather.

In this chapter, we consider additional variants in which there is either a penalty incurred for waiting, or there is a limit on the total time spent waiting, at a given subset of the nodes. We determine the complexity of these additional variants. For variants that are NP-hard, we provide a complexity proof and for variants that are polynomially solvable, we provide an algorithm. We will not discuss the variant in which there is a constraint on waiting at each individual node, which has been shown to be NP-hard, by Omer and Poss (2019b).

The remainder of the chapter is organized as follows. In Section 3.2, we formally introduce the basic variants of the time-dependent shortest path problem as well as the variants with penalties and constraints on waiting, which are the focus of our research. In Section 3.3, we discuss a few natural extensions of the basic variants. In Section 3.4. we present NP-completeness proofs for the (remaining) variants with penalties and constraints on waiting that are hard. In Section 3.5 and Section 3.6. we present polynomial time algorithms for the (remaining) variants with penalties and constraints on waiting that are easy. Finally, in Section 3.7, we summarize the complexity status of all variants with penalties and limits on waiting.

3.2 Problem Description and Preliminaries

3.2.1 Problem Setting Description and Illustration

Given is a directed network $D = (N, A)$, with nodes $N = \{1, \ldots, n\}$, arcs $A \subset N \times N$, a time horizon $[0, T]$ and, for each arc in $(i, j) \in A$, a time-dependent arc travel time function $c_{i,j} : [0, T] \to \mathbb{R}_+$. Also given is an origin node, node 1, and a destination node, node n.

We will use the following example to illustrate the problem variants we consider. The example network is given in Figure 3.1. It has time horizon $[0, 5]$. The piecewise linear travel time functions for each arc is given by its breakpoints in Table 3.1. For example, for $t \in [1, 2]$, $c_{2,3}(t) = 1.82 + (1.51 - 1.82)(t - 1) = 2.13 - 0.31t$. Arc $(3, 4)$ has

no breakpoints between times 2 and 5, indicated by dashes in the table, so for $t \in [2,5]$, $c_{3,4}(t) = 0.83 + ((1.00-0.83)/(5-2))(t-2) = (215+17t)/300$. Observe that the slopes of all linear pieces are greater than -1, which implies that the FIFO property is satisfied strictly.

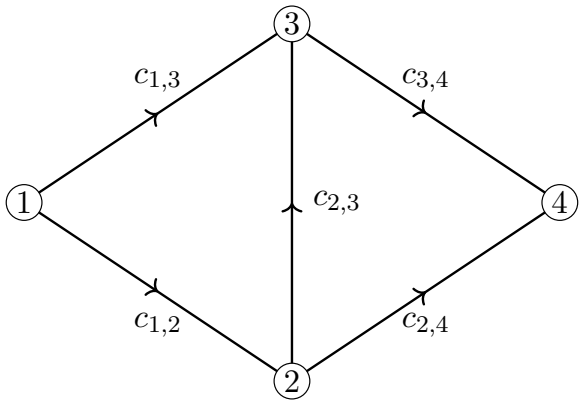


Figure 3.1: Example Network 1.

Table 3.1: Arc travel times at each BP.

BP Time	Arc Travel Times				
	(1,2)	(1,3)	(2,3)	(2,4)	(3,4)
0	1.34	2.85	1.99	1.29	0.61
1	0.66	2.95	1.82	1.02	0.73
2	0.14	3.00	1.51	1.63	0.83
3	0.01	2.98	1.10	2.57	—
4	0.35	2.90	0.67	3.00	—
5	1.00	2.76	0.30	2.54	1.00

In this example, the earliest arrival time at node 2 is 1.34, achieved by departing on arc $(1,2)$ at time $t = 0$. Now the time needed to traverse arc $(2,3)$ if departing at time $t = 1.34$ is $c_{2,3}(1.34) = 2.13 - 0.31 \times 1.34 = 1.7146$, so travel to node 3 on the path consisting of arcs $(1,2)$ and $(2,3)$, without any waiting at either node 1 or node 2, arrives at node 3 at time $1.34+1.7146 = 3.0546$. This is later than $c_{1,3}(0) = 2.85$, so the earliest arrival at node 3 is 2.85, achieved by departing on arc $(1,3)$ at time $t = 0$. Thus the earliest arrival time at node 4 by a path through node 3 must be at time $2.85 + c_{3,4}(2.85) > 2.85$, while the earliest arrival at node 4 via arc $(2,4)$ must be $1.34 + c_{2,4}(1.34) = 1.34 + 1.2274 = 2.5674 < 2.85$.

Thus the solution to the MATP in this example is the path consisting of arcs $(1, 2)$ and $(2, 4)$, departing the origin, node 1, at time 0 and arriving at the destination, node 4, at time 2.5674. This solution is shown in time-expanded form in Figure 3.2, drawn in black (times shown are rounded to two decimal places).

The same sequence of arcs also yields the MDP solution in this example, if departure at the origin is delayed by 2 units of time. At time $t = 2$, travel time on arc $(1, 2)$ is 0.14, and at time $t = 2.14$, travel time on arc $(2, 4)$ is 1.76, giving an arrival time of $2.14 + 1.76 = 3.90$, but a duration of only $3.9 - 2 = 0.14 + 1.76 = 1.90$. In this example, this is also the minimum travel time, so the same arc sequence and departure times on each arc solves the MTTP.

3.2.2 New Problem Variants

The two new variants of time-dependent shortest path problems that we consider both seek a *timed path*, which consists of a path in the network and a specified departure time for each arc in the path, that departs from node 1 at time 0 or later, and arrives at node n no later than time T. If $(i, j), (j, k) \in A$ are arcs used consecutively in a timed path, P, with associated departure times t and t', respectively, then we call $\omega(P, j) := t' - (t + c_{i,j}(t))$ the *waiting time* of P at node j. Naturally, the waiting time at any node in a timed path must be non-negative, and if it is positive, we say the path *waits at* the node. If the timed path departs the origin node, 1, at time t, we say that the waiting time at node 1 is t, while if it arrives at the destination node, n, at time t, we say the waiting time at node n is $T - t$. A timed path, P, has an associated *travel time*, denoted by $\tau(P)$, which is the sum of the traversal times of its arcs at their specified departure times. Thus, the sum of the waiting time at *all* nodes in a timed path, added to its travel time, yields T, the length of the time horizon.

In the illustrative example, the timed path consisting of arc $(1, 2)$ departing at time 2 and arc $(2, 4)$ departing at time 2.14, which solves both the MDP and MTTP in this instance,

has travel time 1.90 (which, in this case, is also its duration). It has waiting time of 2 at node 1, zero at node 2 and a waiting time of $(5 - 3.9) = 1.1$ at node 4. Another example is the timed path consisting of arc $(1, 2)$ departing at time 2 and arc $(2, 4)$ departing at time 2.5. Since $c_{2,4}(2.5) = 2.1$, this timed path has travel time $0.14 + 2.1 = 2.24$. Its waiting time at node 1 is 2, at node 2 is $2.5 - 2.14 = 0.36$ and at node 4 is $5 - (2.5 + 2.1) = 0.4$.

In both problem variants we consider, we *account for* or *tally* waiting at a given set of nodes $M \subseteq N$. We call M the *tally set* and define the *tallied waiting time* of a timed path P to be

$$\omega(P) := \sum_{j \in M} \omega(P, j).$$

For a node $j \notin M$, we say waiting at j is *free*.

Variant with a penalty on tallied waiting time

The first variant we consider applies a penalty to the tallied waiting time. It seeks a timed path, P, so as to minimize $\tau(P) + \alpha\omega(P)$ for some penalty parameter $\alpha \geq 0$. Here $\alpha < 1$ indicates that waiting is "cheaper" than traveling and $\alpha > 1$ indicates that waiting is "more expensive" than traveling. We refer to this variant as the *Time-Dependent Shortest Path Problem with Penalized Waiting* (TDSPP-PW). For nodes in $N \setminus M$, waiting is allowed without penalty.

To illustrate, consider the TDSPP-PW for the example with $\alpha = 0.3$ and $M = \{1\}$. The timed path consisting of arc $(1, 2)$ departing at time 2 and arc $(2, 4)$ departing at time 2.14, which solves both the MDP and MTTP in this instance, has TDSPP-PW objective value of $1.9 + \alpha \times 2 = 2.5$, since its waiting time at node 1 is 2. The optimal solution to this TDSPP-PW consists of the same arcs, but departs on $(1, 2)$ at time 1, and on $(2, 4)$ at time 1.66. It gives the optimal TDSPP-PW objective value: $(0.66 + 1.42) + \alpha \times 1 = 2.38$.

Solutions to two other TDSPP-PW problems using the same network and travel time functions are shown in Figure 3.2, presented in a *Time-Expanded Network* (TEN). In both cases, the tallied waiting penalty, α, exceeds 1 (any value greater than 1 yields the given

solution). The solid path solves the TDSPP-PW with waiting tallied at all nodes other than the destination. Note that this is precisely the MATP optimal solution. The dashed path solves the TDSPP-PW with waiting tallied at all nodes other than the origin. This is precisely the "reverse MATP" optimal solution: it is the path that departs the origin as late as possible. As we shall prove shortly, these problems are indeed equivalent.

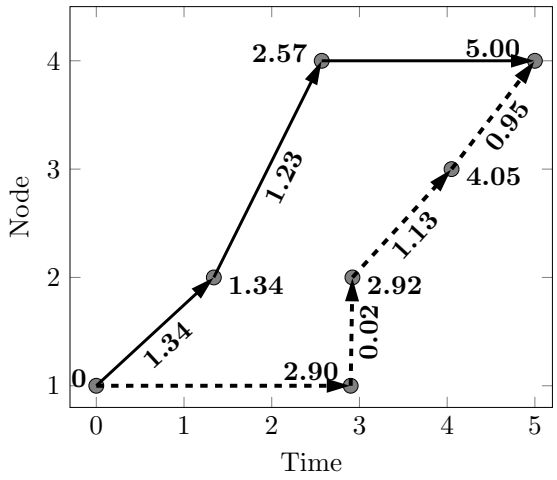


Figure 3.2: The optimal solution to the TDSPP-PW with $\alpha > 1$: the case of $M = N\backslash\{n\} = \{1, 2, 3\}$ is shown as solid lines and the case of $M = N\backslash\{1\} = \{2, 3, 4\}$ is shown as dashed lines.

Variant with a limit on tallied waiting time

The second variant we consider, rather that having a penalty associated with the tallied waiting time, imposes a limit on it. For $W \geq 0$ a given parameter, the *Time-Dependent Shortest Path Problem with Limited Waiting* (TDSPP-LW) seeks timed path P so as to minimize $\tau(P)$ subject to the constraint $\omega(P) \leq W$. For nodes in $N \setminus M$, waiting is unrestricted by the limit, W.

In the illustrative example, taking tally set $M = N \setminus \{n\} = \{1, 2, 3\}$ and waiting limit $W = 0.5$, the optimal TDSPP-LW solution waits at the origin until time 0.5, then departs on arc $(1, 2)$, arriving at time $0.5 + 1.0 = 1.5$, departs immediately on arc $(2, 4)$, (as it must, since it has already reached the waiting time limit), arriving at time $1.5 + 1.325 = 2.825$. The objective value is the path's total travel time: 2.325.

Recovering prior variants as special cases

These two variants provide natural generalizations of the MATP, MDP and MTTP, each of which can be recovered as a special case of TDSPP-PW and TDSPP-LW for a specific choice of M and of α or W, respectively. The MTTP is the case of TDSPP-PW with $\alpha = 0$: irrespective of the choice of M, the waiting penalty is zero, so waiting is, in effect, ignored. Under the FIFO assumption, MATP is the case of TDSPP-LW with $M = N \setminus \{n\}$ and $W = 0$, since in the MATP, waiting at any node other than the destination is suboptimal: this may be enforced as a constraint, in which case minimizing arrival time is equivalent to minimizing total travel time. The MDP, under the FIFO assumption, is the case of TDSPP-LW with $M = N \setminus \{1, n\}$ and $W = 0$, since waiting at any node other than the origin and destination is suboptimal for the MDP, so may be enforced as a constraint, in which case minimizing duration is equivalent to minimizing total travel time under this constraint.

In the remainder of this chapter, we analyze the complexity of the TDSPP-PW and the TDSPP-LW, to determine the effect of the choice of M and of α or W, respectively, on the problem's complexity. We do so under the following assumptions.

3.2.3 Assumptions

The following assumptions on the underlying data apply throughout the remainder of this chapter.

Assumption 1. *The time-dependent arc travel time functions are continuous piecewise linear.*

This is a common assumption in the literature, see, for example, Ichoua et al. (2003) and Figliozzi (2012).

Assumption 2. *The time-dependent travel time functions satisfy the* First-In First-Out *(FIFO) property: for all $(i, j) \in A$ and $t, t' \in [0, T]$ with $t < t'$, $c_{i,j}(t') \geq c_{i,j}(t)$, guaranteeing that a later departure implies a no earlier arrival.*

For continuous piecewise linear functions, this is equivalent to the requirement that the slope of all linear pieces is at least -1. For simplicity of exposition, we require that the FIFO property holds *strictly* (slopes of all linear pieces are strictly greater than -1). Only minor modifications need to be made to the proofs in this chapter to account for the non-strict FIFO property, mainly to deal carefully with the case of multiple optimal solutions.

Assumption 3. *For each node $i \in N \setminus \{1, n\}$, there exists a path from node 1 starting at time 0 visiting node i and reaching node n at or before time T.*

If there exists a node i for which Assumption 3 does not hold, it can be safely removed from the network, as it cannot be used in any feasible solution.

3.3 Further Cases Equivalent to MATP, MDP or MTTP

In Subsection 3.2.1, we noted that specific choices of M and of the penalty/waiting limit parameters for TDSPP-PW/TDSPP-LW immediately results in the problem becoming equivalent to an MTTP, MATP or MDP (where the latter two cases used the FIFO assumption). Hence these parameter choices lead to problems that are solvable in polynomial time. Here we provide arguments showing that other choices also give rise to one of these previously-studied variants.

We begin by observing that there is a symmetry between the choice of M with $n \in M \subseteq N \setminus \{1\}$ and $1 \in M \subseteq N \setminus \{n\}$, i.e., for a problem with one of these choices, there is an equivalent problem with the other. To establish this, it is helpful to first formally define a timed path. For later convenience, we include in the notation for a timed path not only the departure time on each arc of the path, but also redundant information: the arrival time at the head node of each arc on the path.

Definition 12. *A* timed path $P = ((i_1, t_1), (i_2, t_2), \ldots, (i_K, t_K))$ *consists of a sequence of K node and time pairs such that, for all $k = 1, \ldots, K-1$, either $i_k = i_{k+1}$ and $t_{k+1} > t_k$ or $(i_k, i_{k+1}) \in A$ and $t_{k+1} = t_k + c_{i_k, i_{k+1}}(t_k)$. Furthermore, this description is minimal*

in the sense that at most two of the nodes i_{k-1}, i_k and i_{k+1} may be identical, for any $k = 2, \ldots, K-1$.

We say that P starts at node i_1 and ends at node i_K, starting at time t_1 and ending at time t_K. We may also say that P is a timed path from i_1 to i_K. If $i_k \neq i_{k-1}$ for all $k = 2, \ldots, K$, then we say that P is *waiting-free*. P is *contained in* any time interval $[t^-, t^+]$ with $t^- \leq t_1$ and $t_K \leq t^+$; the interval *contains* P. We use $N(P) = \{i_1, i_2, \ldots, i_K\}$ to denote the set of nodes in P. The travel time and tallied waiting time of P are given by

$$\tau(P) = \sum_{\substack{k=1,\\ i_k \neq i_{k+1}}}^{K-1} c_{i_k, i_{k+1}}(t_k) \qquad \text{and} \qquad \omega(P) = \sum_{\substack{k=1,\\ i_k = i_{k+1} \in M}}^{K-1} (t_{k+1} - t_k).$$

Recall that if $i_k = i_{k+1}$ then $\omega(P, i_k) = t_{k+1} - t_k$. We also define $N^w(P) = \{i \in N(P) : \omega(P, i) > 0\}$ to be the set of nodes at which the path waits. So the tallied waiting time of P may be equivalently written as

$$\omega(P) = \sum_{\substack{k=1,\\ i_k = i_{k+1} \in M}}^{K-1} \omega(P, i_k) = \sum_{i \in N^w(P) \cap M} \omega(P, i).$$

For convenience, we require that timed path $P = ((i_1, t_1), \ldots, (i_K, t_K))$ is feasible for the problems we consider only if $i_1 = 1$, $t_1 = 0$, $i_K = n$ and $t_K = T$. To illustrate using the example, the timed path $P = ((1, 0), (1, 2), (2, 2.14), (2, 2.5), (4, 4.6), (4, 5))$ is feasible for the TDSPP-PW and has $\tau(P) = 0.14 + 2.1 = 2.24$. For $M = \{1, 2\}$, it has $\omega(P) = 2 + 0.36 = 2.36$.

Proposition 4. *Given a TDSPP-PW or TDSPP-LW instance with network (N, A), origin node $o \in N$, destination node $d \in N$, travel time function c_a over time horizon $[0, T]$ for each $a \in A$ and tally set $M \subseteq N$, there exists an equivalent TDSPP-PW or TDSPP-LW problem with the same node set, N, and the same tally set, M, but with the origin and destination nodes swapped: it has origin node d and destination node o. Here, we say that*

two problems are equivalent if they have a bijective mapping between the solutions which does not change the solution objective value.

Proof. Consider a given instance (of either the TDSPP-PW or TDSPP-LW) as described in the proposition. Construct another instance by swapping the roles of 1 and n and of 0 and T, and reversing the direction of the arcs, to create arc set $\overleftarrow{A} := \{(j,i) : (i,j) \in A\}$. Construct the travel time function on each reversed arc, $(j,i) \in \overleftarrow{A}$, by $\overleftarrow{c}_{j,i}(s) := c_{i,j}(t)$ where t solves $T - t - c_{i,j}(t) = s$, for all $s \in [0,T]$. We claim that the instance having network $(N, \overleftarrow{A})$, origin node d, destination node o, travel time function $\overleftarrow{c}_a$ over time horizon $[0,T]$ for each $a \in \overleftarrow{A}$ and tally set $M \subseteq N$, is equivalent to the original instance.

To prove this claim, we consider an arbitrary timed path from o to d for the original instance. Say $P = ((i_1,t_1),(i_2,t_2),\ldots,(i_K,t_K))$ from $i_1 = o$ to $i_K = d$ is a timed path for the original instance. This corresponds to a timed path $\overleftarrow{P} = ((i_K, T - t_K),(i_{K-1}, T - t_{K-1}),\ldots,(i_1, T - t_1))$ from d to o for the new instance. For $i_{k+1} \neq i_k$, so $(i_{k+1}, i_k) \in \overleftarrow{A}$, the travel time $\overleftarrow{c}_{i_{k+1},i_k}(T - t_{k+1}) := c_{i_k,i_{k+1}}(t)$ where t solves $T - t - c_{i_k,i_{k+1}}(t) = T - t_{k+1}$, which is equivalent to $t_{k+1} = t - c_{i_k,i_{k+1}}(t)$. By the definition of P as a timed path, this is solved by $t = t_k$. Thus the two paths have identical travel time. Furthermore, for $i_{k+1} \neq i_k$, we have that $(T - t_k) - (T - t_{k+1}) = t_{k+1} - t_k$, so the two paths also have identical waiting time at every node and hence identical tallied waiting time. The result follows. □

Applying this result to the TDSPP-LW with $M = N \setminus \{1\}$, we obtain the following.

Corollary 2. *The case of TDSPP-LW with $M = N \setminus \{1\}$ and $W = 0$ is equivalent to the MATP.*

Proof. By Proposition 4, any instance of TDSPP-LW with tally set $N \setminus \{1\}$ and $W = 0$ has an equivalent instance of TDSPP-LW with tally set $N \setminus \{n\}$ and $W = 0$. This variant is, as discussed in Subsection 3.2.1, equivalent to the MATP. □

That the TDSPP-PW with $M = N$ and $0 < \alpha \leq 1$ can be solved in polynomial time follows from the following straightforward observation that it is equivalent to solving an

MTTP.

Proposition 5. *The TDSPP-PW with $M = N$ and $0 < \alpha \leq 1$ is equivalent to the MTTP.*

Proof. The time in the planning horizon must be divided between travel time and waiting time, and since waiting costs no more than traveling, minimizing the combined objective is equivalent to minimizing travel time. More formally, when $M = N$, waiting time is tallied at every node, including the origin and destination, so for any timed path P from 1 to n starting and ending within the horizon $[0, T]$, it must be that $\tau(P) + \omega(P) = T$. Thus the problem of minimizing $\tau(P) + \alpha\omega(P) = \tau(P) + \alpha(T - \tau(P)) = \alpha T + (1 - \alpha)\tau(P)$ is equivalent to minimizing $\tau(P)$ when $1 - \alpha \geq 0$. □

A critical concept used to prove some of the results that follow is that a timed path can be replaced with one that follows the same sequence of arcs and waits for the same amount of time at each node, but departs earlier (or arrives later) while taking "not much more" travel time. This concept is formalized in the following lemma.

Lemma 2. *Let P be a timed path starting at node i at time t_i and ending at node j at time t_j, that does not wait at either i or j. Let $[t^-, t^+]$ be a time interval that contains $[t_i, t_j]$. Then for any $\beta \in (0, t_i - t^-]$, the path Q that*

1. *has the same arc (and node) sequence as P,*
2. *starts at node i at time $t_i - \beta \geq t^-$, and*
3. *spends the same amount of time waiting at nodes as P does and so has $\omega(Q, h) = \omega(P, h)$ for all $h \in N(P)$,*

satisfies $\tau(Q) < \tau(P) + \beta$. Similarly, for all $\beta \in (0, t^+ - t_j]$, the path Q that

1. *has the same arc (and node) sequence as P,*
2. *ends at node j at time $t_j + \beta \leq t^+$, and*

3. *has* $\omega(Q,h) = \omega(P,h)$ *for all* $h \in N(P)$,

satisfies $\tau(Q) < \tau(P) + \beta$.

Proof. Consider some $\beta \in (0, t_i - t^-]$ and define Q to be the timed path that starts at node i at time $t_i - \beta$, follows the same arc sequence as P does, waiting at each node for the same amount of time as P. By inductive application of the (strict) FIFO property, Q ends at node j before P does. Say Q ends at time $t' < t_j$. Thus, since

$$\tau(Q) + \sum_{h \in N(Q)} \omega(Q,h) = t' - (t_i - \beta) \text{ and } \tau(P) + \sum_{h \in N(P)} \omega(P,h) = t_j - t_i,$$

it must be that

$$\begin{aligned}
\tau(Q) &= t' - t_i + \beta - \sum_{h \in N(Q)} \omega(Q,h) \\
&= t' - t_i + \beta - \sum_{h \in N(P)} \omega(P,h) \\
&< t_j - t_i - \sum_{h \in N(P)} \omega(P,h) + \beta \\
&= \tau(P) + \beta,
\end{aligned}$$

as required.

The case of $\beta \in (0, t^+ - t_j]$ with Q ending at node j at time $t_j + \beta \leq t^+$ is similar; FIFO is applied inductively in the reverse direction, backward along the path from j. □

The proof of Lemma 2 employs the observation that (strict) FIFO extends to timed paths that wait the same amount of time at each node, by inductive application of the (strict) FIFO property. Since this observation is useful in later proofs, we state it formally.

Observation 1. *If two timed paths* P_1 *and* P_2 *have the same node sequence and wait at the same nodes for the same amount of time, but* P_1 *departs earlier than* P_2*, then the timed*

copy of each node in the sequence for P_1 is earlier than the timed copy for that same node in P_2.

We now give results for the TDSPP-PW with $\alpha > 1$.

Lemma 3. *For the TDSPP-PW, when $\alpha > 1$ it is suboptimal to wait at nodes in the tally set if either the origin node or the destination node is not in the tally set.*

Proof. Suppose, for contradiction, that P is an optimal timed path that waits at a node in the tally set, M. Let $i \in M$ be such a node, and say (i, t) and (i, t') are two consecutive elements in P with $t' > t$. Split P into two timed paths: P_1 is the part of P starting from node 1 and ending with (i, t) and P_2 is the part of P starting with (i, t') and ending at node n. Consider the case that $1 \notin M$. Then we may, without loss of generality, assume that P, and hence P_1, has no waiting at node 1. Note P_1 is contained in the interval $[0, t']$. Applying the second part of Lemma 2 to P_1 with $\beta = t' - t = \omega(P, i)$, there exists a timed path Q from node 1 to node i that arrives at time t' with $\omega(Q) = \omega(P_1)$ and $\tau(Q) < \tau(P_1) + \beta$. By concatenating Q and P_2, noting that the resulting timed path has no waiting at node $i \in M$, we obtain a timed path with tallied waiting time equal to that of P minus $\omega(P, i)$. Thus, by writing $Q \cup P_2$ to denote the concatenation of Q and P_2, we have that

$$\begin{aligned}\tau(Q \cup P_2) + \alpha\omega(Q \cup P_2) &= \tau(Q) + \tau(P_2) + \alpha(\omega(P) - \omega(P, i)) \\ &< \tau(P_1) + \beta + \tau(P_2) + \alpha(\omega(P) - \beta) = \tau(P) + \alpha\omega(P) - (\alpha - 1)\beta,\end{aligned}$$

which contradicts the optimality of P, since $\alpha - 1 > 0$. The alternative case, of $n \notin M$, can be shown similarly to result in a contradiction. In this case, the second part of Lemma 2 is applied to P_2, which is contained in the interval $[t, T]$, again applied with $\beta = t' - t > 0$. □

Proposition 6. *The case of TDSPP-PW where $M = N \setminus \{1, n\}$ and $\alpha > 1$ is equivalent to MDP.*

Proof. By Lemma 3, any optimal solution to the TDSPP-PW in this case cannot wait at any node in M. Therefore, this TDSPP-PW is equivalent to minimizing travel time subject to no waiting at any node except the origin or destination, which is exactly the MDP (under FIFO). □

Proposition 7. *The case of TDSPP-PW where $M = N \setminus \{n\}$ or $M = N \setminus \{1\}$, and $\alpha > 1$, is equivalent to MATP.*

Proof. Suppose $\alpha > 1$. By Lemma 3, any optimal solution to the TDSPP-PW cannot wait at any node in M. Therefore, the TDSPP-PW with $M = N \setminus \{n\}$ is equivalent to minimizing travel time subject to no waiting at any node except the destination, which is exactly the MATP (under FIFO). The case of $M = N \setminus \{1\}$ follows by equivalence to the $M = N \setminus \{n\}$ case, from Corollary 2. □

We have so far established that for specific choices of M and specific values of α and W, we recover MATP, MDP and MTTP. Hence, for these choices, the TDSPP-PW and TDSPP-LW are solvable in polynomial time. Table 3.3 and Table 3.2 shows the cases for which the complexity status has been established so far.

Table 3.2: Complexity results for variants of TDSPP-PW obtained so far. Recall that the case of the TDSPP-PW with $\alpha = 0$ is precisely the MTTP, irrespective of the choice of M.

	$0 < \alpha \leq 1$	$\alpha > 1$
$M = N$	Polytime (Proposition 5)	
$N \setminus M = \{1, n\}$		Polytime (Proposition 6)
$N \setminus M = \{n\}$ or $\{1\}$		Polytime (Proposition 7)

Table 3.3: Complexity results for variants of TDSPP-LW obtained so far.

	$W = 0$	$W > 0$
$M = N$		
$N \setminus M = \{1, n\}$	Polytime (MDP)	
$N \setminus M = \{n\}$ or $\{1\}$	Polytime (MATP)	

In what follows, we will provide results that allow us to complete the table, and to exhaustively resolve the complexity for all cases of the tally set M.

3.4 Cases that are NP-Hard

Some variants of TDSPP-PW and TDSPP-LW are generalizations of the *Exact Path Length Problem* (EPLP): given a directed graph $D = (N, A)$ with integer costs on each arc, an integer B and two nodes $i, j \in N$, determine whether there is a path in D from node i to node j with cost exactly B. Without loss of generality, we assume $i = 1$ and $j = n = |N|$.

Theorem 1. *(Nykänen and Ukkonen 2002) The EPLP problem is weakly NP-hard even if the edge weights of the graph G are non-negative integers.*

Theorem 2. *The variant of TDSPP-PW in which waiting is penalized at every node ($M = N$) and the waiting cost is greater than the travel cost ($\alpha > 1$) is weakly NP-hard.*

Proof. We provide a reduction from EPLP. Given an instance of the EPLP as described above, construct an instance of the TDSPP-PW on the same graph, with constant (time-independent) travel times on arcs equal to the cost of the arc in EPLP and with time horizon $[0, B]$. We claim that the optimal value of the TDSPP-PW is B if and only if the answer to EPLP is 'Yes'. To see this, first consider any feasible timed path, P, for the TDSPP-PW. The sum of the EPLP costs on the arcs used in P is precisely its travel time, $\tau(P)$. Since waiting is tallied at every node, $\tau(P) + \omega(P) = B$, and the objective value of P is $\tau(P) + \alpha\omega(P)$. Thus for any choice of $\alpha > 1$, the optimal value of the TDSPP-PW is at least B, and can only equal B if there exists a timed path with no waiting. Thus the optimal value of the TDSPP-PW is B if and only if there is a feasible timed path, P, with $\omega(P) = 0$, which is equivalent to $\tau(P) = B$, and the claim is proved. It is clear that this reduction is polynomial. EPLP is NP-hard by Theorem 1, therefore this variant of TDSPP-PW is NP-hard. □

Theorem 3. *The variant of TDSPP-LW in which waiting is tallied at every node ($M = N$) is weakly NP-hard for $W \geq 0$.*

Proof. We provide a reduction from EPLP to the problem of deciding whether or not the TDSPP-LW problem is feasible. Given an instance of the EPLP as described above, construct an instance of the TDSPP-LW on the same graph, with constant (time-independent) travel times on arcs equal to the cost of the arc in EPLP and with time horizon $[0, B]$. Now for any $W \in [0, 1)$, there exists a feasible solution to the resulting TDSPP-LW instance if and only if the answer to EPLP is 'Yes'. This is because any feasible solution, P, to the TDSPP-LW has $\tau(P) + \omega(P) = B$ and $0 \leq \omega(P) \leq W < 1$. But B and $\tau(B)$ are integers, so $\omega(P)$ is too: it must be that $\omega(P) = 0$ and hence $\tau(P) = B$. Thus the TDSPP-LW instance has a feasible solution if and only if it has a timed path with travel time exactly B. However, the sum of the EPLP costs on the arcs used in a timed path for the TDSPP-LW is precisely its travel time and the claim follows. It is clear that this reduction is polynomial. EPLP is NP-hard by Theorem 1, therefore this variant of TDSPP-LW is NP-hard. □

Next, we consider the case of a general tally set M. Let $\{a_1, a_2, \ldots, a_n\}$ be a set of integers, $I = \{1, \ldots, n\}$, $\sum_{j \in I} a_j = 2B$, and $b_i := \sum_{j=1}^{i-1} 3a_j$ for $i = 1, \ldots, n$ (taking $b_1 := 0$). Construct an instance of TDSPP-LW as follows:

- nodes $N = \{1, 2, 3, \ldots, 2n-1, 2n, 2n+1\}$, with origin 1 and destination $2n+1$;
- arcs $A = \bigcup_{i=1,2,\ldots,n} \{(2i-1, 2i), (2i-1, 2i+1), (2i, 2i+1)\}$;
- tally set $M = \{2i \mid i = 1, 2, \ldots, n\}$;
- arc travel time functions given by
  - $c_{2i-1,2i}(t) = |b_i - t|$, for all $i = 1, 2, \ldots, n$,
  - $c_{2i,2i+1}(t) = |b_{i+1} - t|$, for all $i = 1, 2, \ldots, n$ and
  - $c_{2i-1,2i+1}(t) = |b_i - t| + a_i$, for all $i = 1, 2, \ldots, n$, for all $t \in \mathbb{R}$;
- time horizon $[0, T]$ with $T = 4B$; and
- tallied waiting time limit $W = 3B$.

The network is shown in Figure 3.3 and the travel time functions are shown in Figure 3.4.

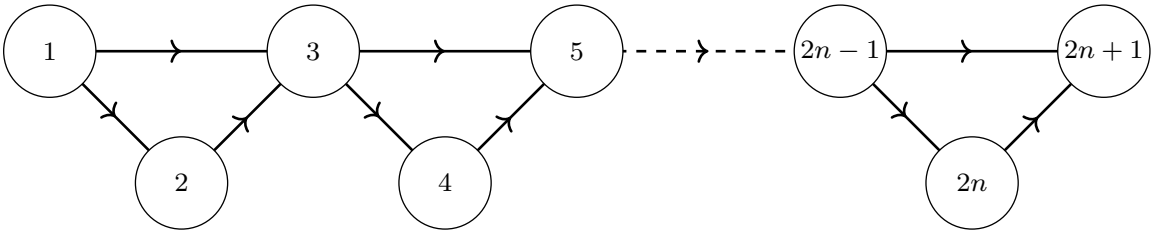
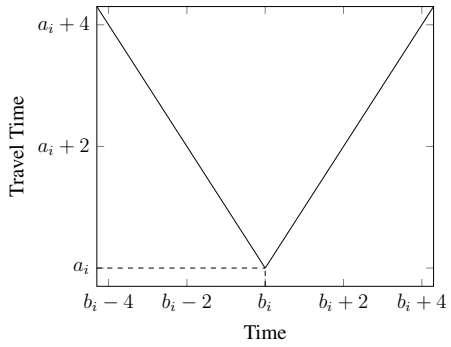
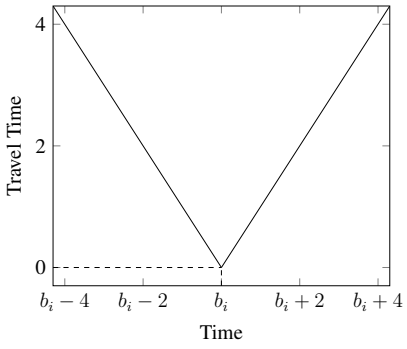


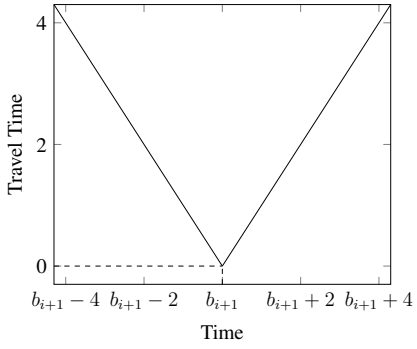
Figure 3.3: Example Network 2.



(a) Travel time function $c_{2i-1,2i+1}$



(b) Travel time function $c_{2i-1,2i}$



(c) Travel time function $c_{2i,2i+1}$

Figure 3.4: Travel time functions for Example Network 2.

For this instance, all travel time function gradients are in $\{-1, 1\}$, so the FIFO property is satisfied, but not strictly, Furthermore, the travel time functions are nonnegative at all times, but do not satisfy the positive travel time assumption that we usually require. The instance can be modified slightly to satisfy these properties, and the arguments below can be adapted to the modified instance. For simplicity, we use the instance without these modifications.

The network $D = (N, A)$ comprises (overlapping) subnetworks $D_i = (N_i, A_i)$, where $N_i = \{2i-1, 2i, 2i+1\}$ and $A_i = \{(2i-1, 2i), (2i-1, 2i+1), (2i, 2i+1)\}$, for $i = 1, 2, \ldots, n$. To traverse D_i from node $2i-1$ to node $2i+1$, there are two options: either travel along path $p_i = (2i-1, 2i+1)$, using only odd numbered nodes, or along path $q_i = (2i-1, 2i, 2i+1)$, using one even numbered node.

For a timed path P from node 1 to node $2n+1$, we define $f(P) = \tau(P) + \frac{1}{3}\omega(P)$. Let P_i denote the part of P restricted to D_i. Thus P_i is a timed path using either the arc sequence of p_i or the arc sequence of q_i. Let $f_i(P)$ denote the contribution to $f(P)$ of the subpath in D_i, which must be either $\tau(P_i)$ if P_i uses p_i or $\tau(P_i) + \frac{1}{3}\omega(P_i) = \tau(P_i) + \frac{1}{3}\omega(P_i, 2i)$ if P_i uses q_i, since the only node in N_i that is in the tally set is node $2i$. Nodes with an odd index are not in the tally set M, thus waiting at odd indexed nodes does not need to be accounted for and hence $f(P) = \sum_{i=1}^{n} f_i(P)$.

Lemma 4. *For any $i \in I$ and any timed path S in D_i from node $2i-1$ to node $2i+1$, $f(S) \geq a_i$. Moreover, $f(S) = a_i$ only when (i) S departs node $2i-1$ at time b_i and (ii) if S uses q_i then S departs node $2i$ at time $b_i + 3a_i$.*

Proof. Let $i \in I$ and S be a timed path in D_i from node $2i-1$ to node $2i+1$. If S uses p_i, then since travel time function $c_{2i-1,2i+1}(t) \geq a_i$ for all t, we have $f(S) \geq a_i$. Moreover, $c_{2i-1,2i+1}(t) = a_i$ if and only if $t = b_i$, and the result follows.

Otherwise, it must be that S uses q_i. Consider the timed path $S^* = ((2i-1, b_i), (2i, b_i), (2i, b_i + 3a_i), (2_i + 1, b_i + 3a_i))$, which departs at node $2i-1$ at time b_i and departs node $2i$ at time $b_{i+1} = b_i + 3a_i$, has travel time $\tau(S^*) = 0$ and waiting time $\omega(S^*) = 3a_i$, and, thus, $f(S^*) = a_i$. We now show that this is a local minimum of f for timed paths using q_i.

Observe that departing earlier at node $2i-1$ or departing later at node $2i$ results in an increase in travel time and an increase in waiting time at node $2i$, and so cannot lead to a reduction in the value of f. Also, departing node $2i-1$ at time $b_i + \delta_1$ for small $\delta_1 > 0$ results in an increase in travel time of δ_1 and a decrease in waiting time at node $2i$ of $2\delta_1$,

giving a change in $f(S^*)$ of $\delta_1 - \frac{2}{3}\delta_1 = \frac{1}{3}\delta_1 > 0$. Departing node $2i$ at time $b_i + 3a_i - \delta_2$ for small $\delta_2 > 0$ results in an increase in travel time of δ_2 and a decrease in waiting time at node $2i$ of δ_2, giving a change in $f(S^*)$ of $\delta_2 - \frac{1}{3}\delta_2 = \frac{2}{3}\delta_2 > 0$. Due to the positive waiting time at node $2i$ in S^*, small changes in these departure times can be carried out independently and the impact on f is additive. Thus S^* is a strict local minimizer of f for timed paths using q_i.

It is clear that f for any timed path using q_i is a convex function of $(t_1, t_2) \in \mathbb{R}^2$ where t_1 is the departure time at node $2i - 1$ and t_2 is the departure time at node $2i$. Hence S^* must be a unique global minimum. □

The result below follows directly from Lemma 4, the definition of the travel time functions and the definition of b_{i+1}, which gives $b_{i+1} = b_i + 3a_i$.

Corollary 3. *For any $i \in \{1, \ldots, n\}$ and any timed path S in D_i from node $2i - 1$ to node $2i + 1$, if $f(S) = a_i$ then (i) S arrives at node $2i + 1$ no later than b_{i+1}, (ii) if S uses p_i then $\tau(S) = a_i$ and $\omega(S) = 0$, and (iii) if S uses q_i then $\tau(S) = 0$ and $\omega(S) = 3a_i$.*

Lemma 5. *Any feasible timed path P with $\tau(P) \leq B$ satisfies $f(P) \leq 2B$.*

Proof. A feasible timed path P has $\omega(P) \leq 3B$. Hence, $f(P) = \tau(P) + \frac{1}{3}\omega(P) \leq 2B$. □

Lemma 6. *Any timed path P has $f(P) \geq 2B$. Moreover, $f(P) = 2B$ only when for all $i \in I$, P_i departs node $2i - 1$ at time b_i and if P_i uses q_i then P departs node $2i$ at time $b_i + 3a_i$.*

Proof. By Lemma 4, $f(P_i) \geq a_i$ for each $i \in I$. As already noted $f(P) = \sum_{i \in I} f(P_i) \geq \sum_{i \in I} a_i = 2B$. Obviously, if $f(P) = 2B$ then $f(P_i) = a_i$ for all $i \in I$, and Lemma 4 gives the conditions for this to hold. □

Theorem 4. *The variant of* TDSPP-LW *in which waiting is constrained at a subset of nodes $M \subseteq N$ and there is a positive limit on total waiting ($W > 0$) is NP-hard.*

Proof. We prove that its decision version, in which we want decide if there is path with travel time less than a given constant, is NP-Complete by a transformation from PARTITION.

PARTITION: Given a set of integers $\{a_1, a_2, \ldots, a_n\}$, does there exist a subset $S \subseteq I = \{1, \ldots, n\}$ such that $\sum_{j \in S} a_j = \frac{1}{2} \sum_{j \in I} a_j = B$?

The transformation takes an instance of PARTITION and constructs an instance of TDSPP-LW as outlined at the start of this section and takes the constant to be B, i.e., we want to decide if there is a feasible timed path with travel time less than or equal to B.

First, we show that a YES instance of PARTITION induces a YES instance of TDSPP-LW. Let $S \subset I$ with $\sum_{i \in S} a_i = B$. Take the timed path P formed by concatenating, for $i = 1, \ldots, n$, the timed path that uses p_i whenever $i \in S$ and q_i whenever $i \notin S$, with the departure times specified in Lemma 4. Such concatenation is feasible by Corollary 3. By construction, $\tau(P) = B$ and $\omega(P) = 3B$.

Next, we show that a YES instance of TDSPP-LW with constant B induces a YES instance of PARTITION. Lemma 5 and Lemma 6 imply that a feasible timed path P with $\tau(P) \leq B$ has $f(P) = 2B$. Now since the tallied waiting time limit $W = 3B$, it must be that $\omega(P) \leq 3B$, so

$$f(P) = \tau(P) + \tfrac{1}{3}\omega(P) = 2B \;\; \Rightarrow \;\; \tau(P) + \tfrac{1}{3}(3B) \geq 2B \;\; \Rightarrow \;\; \tau(P) \geq B.$$

Hence $\tau(P) = B$ and $\omega(P) = 3B$. Furthermore, since equality holds, by Lemma 6 and Corollary 3, $\tau(P) = \sum_{i \in I : P_i \text{ uses } p_i} \tau(P_i)$. Thus selecting i to be in S whenever P_i uses p_i is chosen (and not selecting i to be in S whenever q_i is used), a feasible solution to the instance of PARTITION is obtained, since

$$B = \tau(P) = \sum_{i \in I : P_i \text{ uses } p_i} \tau(P_i) = \sum_{i \in S} a_i,$$

by construction. □

Because neither 1 nor n are in the tally set M in the transformation used to prove Theorem 4, we have the following theorem as an immediate consequence.

Theorem 5. *The variant of* TDSPP-LW *in which waiting is constrained at a subset of nodes $M \subseteq N$ with $1, n \notin M$ and a positive limit on total waiting ($W > 0$) is NP-hard.*

3.5 Cases of TDSPP-PW Solvable in Polynomial Time via a Time-Expanded Network

We now discuss cases that can be reduced to solving a (standard) shortest path problem in a time-expanded network that has size polynomial in the size of the given instance. We first explain how the time-expanded network is constructed.

3.5.1 The Time-Expanded Network

The time-expanded networks central to our polynomial time complexity results are constructed as a finite set of *timed nodes*, each a node-time pair of the form (i, t) with $i \in N$ and $t \in [0, T]$, and *timed arcs*, of the form $((i, t), (j, t'))$ where (i, t) and (j, t') are timed nodes and $t' \geq t$. We also refer to a timed node (i, t) as a *timed copy of* i. The timed node set is constructed by solving one MATP and one reverse MATP (introduced in Subsection 3.2.2) for each breakpoint of an arc travel time function, as follows. For arc a outgoing from node i and time t a breakpoint of its travel time function:

- (i, t) is included in the set of timed nodes,
- for each $j \in N \setminus \{i\}$ reachable from (i, t) within the time horizon, (j, t') is included in the set of timed nodes, where t' is the earliest arrival time at j if departing i at time t (the value of the MATP with origin node i, destination node j and time horizon $[t, T]$), and

- for each $j \in N \setminus \{i\}$ with i reachable from $(j, 0)$ no later than t, (j, t') is included in the set of timed nodes, where t' is the latest departure time from j to arrive at node i at time t (the value of the reverse MATP with origin node j, destination node i and time horizon $[0, t]$).

Algorithms for the MATP, described in Cooke and Halsey (1966), Orda and Rom (1990), Dean (2004b), for example, can easily be adapted to find the minimum arrival time path to *all* nodes in $N \setminus \{i\}$ for the same computational effort as finding the minimum arrival time path to just one destination. Such algorithms ensure that if the minimum arrival time path departing from node i at time t arrives at node k at time t'' along arc (j, k), having departed j at time t', then t' is the minimum arrival time at j starting from (i, t). Thus such algorithms generate a timed version of a forward shortest path tree, with at most one timed node per node in the network. As per Proposition 4, reverse MATP is equivalent to MATP; solving reverse MATP generates a timed version of a backward shortest path tree. Hence, the number of timed nodes in the time-expanded network is at most $2nK$, where K is the total number of travel time function breakpoints.

The set of timed arcs is constructed by including $((i, t), (j, t'))$ whenever arc $(i, j) \in A$ is traversed starting at time t in any such forward or backward shortest path tree. In this case it must be that $t' = t + c_{i,j}(t)$. We call such timed arcs *travel arcs*. We say that the *travel time* of timed arc $((i, t), (j, t'))$ is $c_{i,j}(t) = t' - t$. In addition, the set of timed arcs also includes *waiting arcs*: if $(i, t_1), (i, t_2), \ldots, (i, t_r)$ is the set of all timed copies of i, listed in increasing chronological order, then $((i, t_{s-1}), (i, t_s))$, which is referred to as a waiting arc, is included in the set of timed arcs, for each $s = 2, \ldots, r$.

Clearly any path in the TEN corresponds to a timed path in the original network, which can be expressed simply as the timed node sequence in the TEN path, omitting intermediate timed copies of the same node whenever more than two appear consecutively.

In what follows, we will show that for all remaining cases of the TDSPP-PW, there exists an optimal solution to any instance that corresponds to a path in its TEN. Furthermore,

we will be able to construct lengths for the arcs in the TEN so that any shortest path, with respect to these lengths, from $(1, 0)$ to (n, T) in the TEN corresponds to an optimal solution to the TDSPP-PW instance.

3.5.2 Preliminaries

Definition 13. *A* travel subpath *of a given timed path is a maximal consecutive sequence of timed nodes in the timed path with no consecutive timed copies of the same node.*

Thus a timed path is the concatenation of a sequence of travel subpaths. Note that a node is a start or end of a travel subpath if and only if the node is 1 and the travel subpath starts with $(1, 0)$, or n and the travel subpath ends with (n, T), or if the node is in $N^w(P)$.

To illustrate, consider the example and the feasible timed path $P = ((1, 0), (1, 2), (2, 2.14),$
$(2, 2.92), (3, 4.05), (4, 5))$. Here $N^w(P) = \{1, 2\}$ with $\omega(P, 1) = 2$ and $\omega(P, 2) = 0.78$, and P has three travel subpaths: $((1, 0))$, $((1, 2), (2, 2.14))$ and $((2, 2.92), (3, 4.05), (4, 5))$.

If a timed path consists of three of more travel subpaths then we say that all but the first and last travel subpaths are *intermediate* subpaths. A travel subpath is, itself, a timed path, and so properties of timed paths and terms used to describe them may also be used for travel subpaths.

A key feature of a travel subpath is whether or not it contains a breakpoint. Note that we consider time 0 to be a breakpoint at node 1 and time T to be a breakpoint at node n, regardless of whether the travel time function for an arc leaving or entering these nodes, respectively, has a breakpoint at these respective times.

Definition 14. *A travel subpath* contains a breakpoint *if it starts with $(1, 0)$, or ends with (n, T), or if there is an arc $(i, j) \in A$ and a time t that is a breakpoint of $c_{i,j}(\cdot)$ for which the timed node (i, t) appears in the travel subpath.*

Our argument that some solution to a TDSPP-PW instance must occur in its TEN involves *shifting* travel subpaths that do not contain a breakpoint. We thus make use of the

following idea.

Definition 15. *A travel subpath* $P(\Delta)$ *is a* Δ-shifting *of travel subpath* $P = ((i_1, t_1), \ldots, (i_K, t_K))$ *if* $P(\Delta) = ((i_1, t'_1), \ldots, (i_k, t'_K))$ *where* $t'_1 = t_1 + \Delta$ *and* $t'_k = t'_{k-1} + c_{i_{k-1},i_k}(t'_{k-1})$ *for* $k = 2, \ldots, K$.

Note that, from Observation 1, for $P(\Delta)$ a Δ-shifting of P, if $\Delta < 0$ then $t'_k < t_k$ for all $k = 1, \ldots, K$, while if $\Delta > 0$ then $t'_k > t_k$ for all $k = 1, \ldots, K$.

As consequence of properties of compositions of piecewise affine functions, if a travel subpath does not contain a breakpoint, then its travel time is a locally affine function of the departure time on its first node.

Observation 2. *If travel subpath* P *does not contain a breakpoint, then there exist* $\varepsilon^-, \varepsilon^+ > 0$ *such that* $\tau(P(\Delta))$ *is an affine function of* Δ *for all* $\Delta \in (-\varepsilon^-, \varepsilon^+)$. *Furthermore, if* $\varepsilon^-, \varepsilon^+$ *are the maximal such values, then both* $P(-\varepsilon^-)$ *and* $P(\varepsilon^+)$ *contain a breakpoint.*

In what follows, when we *shift* P *back to a breakpoint*, we mean that we replace P by $P(-\varepsilon^-)$ where $\varepsilon^- \geq 0$ is the largest value such that $P(\delta)$ does not contain a breakpoint for all $\delta \in (-\varepsilon^-, 0)$. Similarly, we *shift* P *forward to a breakpoint* by replacing P with $P(\varepsilon^+)$ where $\varepsilon^+ \geq 0$ is the largest value such that $P(\delta)$ does not contain a breakpoint for all $\delta \in (0, \varepsilon^+)$.

3.5.3 Complexity Results

In this section, we consider the TDSPP-PW with either waiting cost not more than travel cost ($\alpha \leq 1$) or at least one node at which waiting is not tallied ($M \subset N$). (Recall that the only remaining case of TDSPP-PW, namely that with $\alpha > 1$ and $M = N$, is NP-hard, shown in Theorem 2.) Our approach is structured as follows.

We first show that there is an optimal solution to any TDSPP-PW instance with each of its travel subpaths containing a breakpoint. We then show that if either $\alpha \leq 1$ or there is an optimal solution with zero tallied waiting, then there must be an optimal solution in which

each travel subpath consists of two concatenated MATP solutions to/from a breakpoint. Thus each travel subpath corresponds to a sequence of timed arcs of precisely the form of the travel arcs in the TEN for the instance.

It is then straightforward to prove that solving a TDSPP-PW instance with $\alpha \leq 1$ can be done by solving a shortest path problem in its TEN. The case of $\alpha > 1$ and $M \neq N$ requires some additional results, to establish that there is an optimal solution which has zero tallied waiting. We conclude by showing that any instance of the TDSPP-PW with $\alpha > 1$ and $M \subset N$ can be solved in polynomial time, by solving a shortest path in its associated TEN.

Our first result generalizes that of Foschini et al. (2014) for the MDP.

Proposition 8. *For any instance of TDSPP-PW, there exists an optimal timed path such that each of its travel subpaths includes at least one timed node at a breakpoint. Furthermore, if P is any optimal solution, there exists P', also an optimal solution, with a breakpoint in each of its travel subpaths and with $N^w(P') \subseteq N^w(P)$.*

Proof. Suppose that P is an optimal timed path having a travel subpath, S, that does *not* include a breakpoint. Recall that for P to be feasible, it must start with $(1, 0)$ and end with (n, T), which are considered to be breakpoints. So S must be an intermediate travel subpath. Let τ^- be the end time of the travel subpath immediately preceding S and let τ^+ be the start time of the travel subpath immediately after S in P.

We claim that there is a Δ-shifting of S so that (i) $S(\Delta)$ is contained in $[\tau^-, \tau^+]$, (ii) the path formed by replacing S in P by $S(\Delta)$ is optimal, and (iii) either $S(\Delta)$ contains a breakpoint, or it starts at τ^- or ends at τ^+. In the latter two cases, $S(\Delta)$ is no longer a travel subpath of the new optimal path say, since it is not maximally waiting-free; it concatenates with either the preceding travel subpath or the next travel subpath in P to form a longer travel subpath. Thus the new optimal path has either one more travel subpath containing a breakpoint or it has one fewer travel subpath than the original optimal path, P. This procedure cannot introduce waiting at any node where there was no waiting in

P. Applying this procedure repeatedly must end with an optimal path, P', in which every travel subpath contains a breakpoint, and $N^w(P') \subseteq N^w(P)$, as required. We now prove the claim.

First, apply Observation 2 to travel subpath S, so that $\tau(S(\Delta))$ is affine in Δ, given by $\tau(S(\Delta)) = m\Delta + \tau(S)$ for some $m \in \mathbb{R}$, for all $\Delta \in (-\varepsilon^-, \varepsilon^+)$, with $\varepsilon^-, \varepsilon^+ > 0$ and maximal.

Suppose S starts at node i and ends at node j. For $\Delta \in (-\varepsilon^-, \varepsilon^+)$ such that $S(\Delta)$ is contained in $[\tau^-, \tau^+]$, replacing S with $S(\Delta)$ in P to create a new feasible path, P', will result in

$$\tau(P') = \tau(P) + m\Delta, \;\; \omega(P', i) = \omega(P, i) + \Delta \;\text{ and }\; \omega(P', j) = \omega(P, j) - (1+m)\Delta.$$

Thus

$$\begin{aligned}\tau(P') + \alpha\omega(P') &= \tau(P) + \alpha\omega(P) + \left\{\begin{array}{ll} m\Delta, & \text{if } i, j \notin M, \\ m\Delta + \alpha(\Delta - (1+m)\Delta), & \text{if } i, j \in M \\ m\Delta + \alpha\Delta, & \text{if } i \in M, j \notin M \\ m\Delta - \alpha(1+m)\Delta, & \text{if } i \notin M, j \in M \end{array}\right\}. \\ &= \tau(P) + \alpha\omega(P) + \left\{\begin{array}{ll} m\Delta, & \text{if } i, j \notin M, \\ (1-\alpha)m\Delta, & \text{if } i, j \in M \\ (m+\alpha)\Delta, & \text{if } i \in M, j \notin M \\ (m - \alpha(1+m))\Delta, & \text{if } i \notin M, j \in M. \end{array}\right\}.\end{aligned}$$

In each case, m and α (if relevant) must be such that the coefficient of Δ is zero, otherwise some positive or negative value of Δ creates a new path with better TDSPP-PW objective value than P, which is optimal. In other words, the coefficient of Δ is zero in every case, otherwise Δ can be chosen to contradict optimality of P. Hence $\Delta = -\min\{\varepsilon^-, t - \tau^-\}$ must satisfy the claim, where t is the start time of S. In the case that $\varepsilon^- \leq t - \tau^-$, by

Observation 2, the Δ-shifting of S will contain a breakpoint; otherwise it will start at τ^-. (Note: Δ may equally well be set to ε or the value for which $t + \tau(S(\Delta)) \leq \tau^+$, whichever is smaller. In the latter case, $S(\Delta)$ will end at τ^+.) $\square$

Recall that timed nodes in the TEN are created by solving an MATP starting from each breakpoint, forward, to find the earliest time each node can be reached from the breakpoint, and by solving a reverse MATP from each breakpoint, backward, to find the latest departure time at each node from which the breakpoint can be reached. We call a path solving a forward MATP an *earliest arrival time path* and a path solving a backward MATP a *latest departure time path*. By the FIFO property, these paths do not include any waiting.

Lemma 7. *For any instance of the TDSPP-PW satisfying (i) $\alpha \leq 1$ or (ii) there is an optimal solution, P, with $\omega(P) = 0$, there exists an optimal timed path, P', such that each of its travel subpaths consists of a latest departure time path from a node to a breakpoint concatenated with an earliest arrival time path from the same breakpoint to another node. In case (ii), $\omega(P') = 0$.*

Proof. Consider an instance of the TDSPP-PW satisfying the required conditions. If $\alpha \leq 1$, apply Proposition 8, to obtain an optimal timed path, P such that each of its travel subpaths contains a breakpoint. Otherwise, let P^* be an optimal solution with $\omega(P^*) = 0$, so travel subpaths start or end at nodes $N^w(P^*) \subseteq N(P^*) \setminus M$. Apply Proposition 8, to obtain a new optimal solution, P, with each of its travel subpaths containing a breakpoint and $N^w(P) \subseteq N^w(P^*)$. So $\omega(P) = 0$.

Suppose that S is a travel subpath of P that is *not* the concatenation of a latest arrival path and an earliest arrival path. Suppose S starts with (i, t_i), includes breakpoint (k, t_k) and ends with (j, t_j). Let S^b be a latest departure path from i to (k, t_k), and say it starts with (i, t'_i). By optimality of S^b, it must be that $t'_i \geq t_i$. Let S^f be an earliest arrival path from (k, t_k) to j, and say it ends with (j, t'_j). It must be that $t'_j \leq t_j$. Let P' denote the path formed by replacing S in P by S^b concatenated with S^f, which do not include any waiting.

Then

$$\begin{aligned}\tau(P') &= \tau(P) - (t'_i - t_i) - (t_j - t'_j),\\ \omega(P', i) &= \omega(P, i) + (t'_i - t_i), \text{ and}\\ \omega(P', j) &= \omega(P, j) + (t_j - t'_j).\end{aligned}$$

Now define $\alpha_h = \alpha$ if $h \in M$ and $\alpha_h = 0$ otherwise, for $h = i, j$. By the conditions of the lemma, either $\alpha \leq 1$ or $\omega(P) = 0$, so since $\omega(P, i),\ \omega(P, j) > 0$, it must be that $i, j \notin M$, so $\alpha_i = \alpha_j = 0$. Thus, in every case, $\alpha_i,\ \alpha_j \leq 1$. As a consequence,

$$\begin{aligned}\tau(P') + \alpha\omega(P') &= \tau(P) - (t'_i - t_i) - (t_j - t'_j) + \alpha\omega(P) + \alpha_i(t'_i - t_i) + \alpha_j(t_j - t'_j)\\ &= \tau(P) + \alpha\omega(P) - (1 - \alpha_i)(t'_i - t_i) - (1 - \alpha_j)(t_j - t'_j)\\ &\leq \tau(P) + \alpha\omega(P)\end{aligned}$$

since $t'_i \geq t_i$ and $t'_j \leq t_j$. Since P is optimal for the TDSPP-PW instance, it must be that P' is optimal, too. Furthermore, $N^w(P') = N^w(P)$, so if $\omega(P) = 0$ then $\omega(P') = 0$ too. This procedure can be repeated until an optimal solution satisfying the conditions of the lemma is generated. □

Theorem 6. *The variant of TDSPP-PW in which waiting costs less than traveling, i.e.,* $0 < \alpha \leq 1$*, is solvable in polynomial time.*

Proof. By Lemma 7, there exists an optimal timed path P consisting of a sequence of travel subpaths, each of which is the concatenation of a latest departure time path to a breakpoint and an earliest arrival time path from the same breakpoint. By construction, all timed nodes and timed arcs in these paths are included in the TEN. Furthermore, if a travel subpath of P ends with (i, t) and the travel subpath immediately after it in P starts with (i, t'), then by definition of the travel subpaths, $t' > t$. And by construction of the TEN, there must be a sequence of waiting arcs in the TEN forming a path from (i, t) to (i, t').

Define the length of each travel arc in the TEN to be its travel time. Define the length of each waiting arc in the TEN, of the form $((i,t),(i,t'))$ to be $\alpha(t'-t)$ if $i \in M$ and zero otherwise. Then clearly any path from $(1,0)$ to (n,T) in the TEN corresponds to a feasible timed path for the TDSPP-PW and the length of the TEN path is precisely the corresponding timed path's TDSPP-PW objective value. Thus solving a shortest path problem in the TEN with the given lengths must yield an optimal solution to the TDSPP-PW. The TEN has $\mathcal{O}(2nK)$ nodes, for K the number of breakpoints, and hence the TDSPP-PW can be solved in polynomial time. □

We now turn our attention to the case of TDSPP-PW with waiting cost greater than travel cost. Throughout the remainder of this section, "an instance" means an instance of the TDSPP-PW with $\alpha > 1$ and $M \subset N$ (so $M \neq N$).

We begin by establishing that any optimal solution, P^*, to an instance must have zero tallied waiting, i.e., $\omega(P^*) = 0$ for any optimal timed path P^*. We do this in two steps. First, we show that if every node at which a timed path waits is in the tally set, then the path cannot be optimal.

Lemma 8. *If P is a feasible solution for an instance with the property that $\omega(P,i) = 0$ for all $i \in N(P)$ with $i \notin M$, then either P is waiting-free or P is not optimal.*

Proof. If P is as described, then it must be that $\tau(P) + \omega(P) = T$. Suppose P is not waiting-free. Then it must wait at a node in M, so $\omega(P) > 0$. Thus the TDSPP-PW objective value of P is $\tau(P) + \alpha\omega(P) > T$, since $\alpha > 1$. Now $M \neq N$ so there exists $i \in N \setminus M$. Let S^f be the solution to the MATP with origin 1 and destination i, starting at time 0, giving earliest arrival time at i of t. Let S^b be the solution to the reverse MATP that determines the latest departure time from i so as to arrive at n at time T; let this time be t'. Then, by Assumption 3, $t' \geq t$ and the concatenation of S^f followed by S^b is a feasible timed path; denote this path by P'. By FIFO, the only node at which P' waits is node $i \notin M$. Thus $\omega(P') = 0$ and $\tau(P') + \alpha\omega(P') = \tau(P') \leq T < \tau(P) + \alpha\omega(P)$. Hence P

cannot be optimal. □

We now show that if P^* is an optimal solution that is not waiting-free, then it cannot wait at *any* node in M. The proof makes use of the observation that if a travel subpath of a feasible timed path, P, ends at a node other than the destination, say the travel subpath ends at node $i \neq n$, then the timed path must wait at i, i.e., $\omega(P, i) > 0$. Also, if $\omega(P, i) > 0$ for some feasible P and some node i, it must be that a travel subpath either ends at i, starts at i, or both.

Proposition 9. *If P^* is an optimal solution for an instance, then $\omega(P^*) = 0$.*

Proof. If P^* is waiting-free, then the result follows. Otherwise, suppose, for contradiction, that $\omega(P^*) > 0$, so there exists $i \in M$ with $\omega(P^*, i) > 0$. By Lemma 8, there must exist $j \notin M$ with $\omega(P^*, j) > 0$. There are two cases to consider: either i appears before j in the path, or vice versa. The two cases are symmetric, and the proof for one case easily adapted to the other, so we only give the proof for the case that i appears before j in P^*.

We claim that there must exist a pair of nodes $\hat{i} \in N(P^*) \cap M$ and $\hat{j} \in N(P^*) \setminus M$ so that a travel subpath of P^* starts at $\hat{i}$ and ends at $\hat{j}$, with $\omega(P^*, \hat{i}),\ \omega(P^*, \hat{j}) > 0$. To substantiate the claim, observe that some travel subpath of P^* must start at node i. If the end of that travel subpath is also in M, say it ends at node $i' \in M$, then $i' \neq n$ (since node $j \notin M$ must appear later), so it must be that $\omega(P^*, i') > 0$, and we may replace i with i', which appears later in P^*. This procedure can be repeated until the end of the travel subpath starting at i is *not* in M, say it ends at node $j' \notin M$. Then either $j' = j$ and the claim follows, or $j' \neq n$ since j' must appear before j in P^*. Thus, since j' ends a travel subpath and is not the destination node, it must be that $\omega(P^*, j') > 0$. Taking $\hat{i} = i$ and $\hat{j} = j'$ satisfies the claim.

Let $\hat{i}, \hat{j}$ be as claimed. Let $(\hat{i}, s), (\hat{i}, s')$ with $s' > s$ denote the consecutive node-time pairs in P^* that appear before the consecutive node-time pairs $(\hat{j}, t), (\hat{j}, t')$ with $t' > t$ and let Q denote the travel subpath of P^* that starts with $(\hat{i}, s')$ and ends with $(\hat{j}, t)$. Let

$\Delta = s - s' < 0$ and consider $Q(\Delta)$, the Δ-shifting of Q: it is a waiting-free timed path that starts with $(\hat{i}, s)$ and ends at $\hat{j}$ at some time earlier than t. The path, P, formed by replacing Q in P^* by $Q(\Delta)$ is feasible for the instance. Furthermore, $\tau(Q(\Delta)) < t - s = \tau(Q) + s' - s$ and $\omega(P, i) = 0$. Thus, since $\hat{j} \notin M$, we have that

$$\begin{aligned}
\tau(P) + \alpha\omega(P) &= (\tau(P^*) - \tau(Q) + \tau(Q(\Delta))) + \alpha(\omega(P^*) - \omega(P^*, i) + \omega(P, i)) \\
&< (\tau(P^*) + s' - s) + \alpha(\omega(P^*) - (s' - s)) \\
&= \tau(P^*) + \alpha\omega(P^*) + (1 - \alpha)(s' - s) \\
&< \tau(P^*) + \alpha\omega(P^*)
\end{aligned}$$

since $\alpha > 1$ and $s' < s$. This contradicts the optimality of P^*. □

We are now able to show, as a consequence of the above result, that there is an optimal solution for an instance that has a corresponding path in its TEN. We do so by first establishing that every travel subpath of an optimal solution must solve an MDP over the time interval containing both the travel subpath and any waiting before or after it in the optimal solution.

Proposition 10. *For any instance, there is a path in its TEN that corresponds to an optimal timed path for the instance and that does not use any waiting arcs at nodes in M.*

Proof. Let P be an optimal timed path for the instance with $\omega(P) = 0$, known to exist by Proposition 9. By Lemma 7, there exists another optimal timed path, P', such that every travel subpath consists of a latest departure time path to a breakpoint concatenated with an earliest arrival path from the same breakpoint. Thus every travel subpath is contained in the TEN for the instance, which also includes waiting arcs to link all timed copies of the same node. Since Lemma 7 also guarantees that $\omega(P') = 0$, no waiting arc at a node in M is used. The result follows. □

Theorem 7. *The variant of TDSPP-PW in which waiting is penalized at nodes $M \subset N$ and the waiting cost is greater than the travel cost, i.e., $\alpha > 1$, is solvable in polynomial time.*

Proof. Consider an instance of this variant of TDSPP-PW, and its associated TEN. Remove all waiting arcs at each node in M. Define the length of each travel arc in the TEN to be its travel time. Define the length of each (remaining) waiting arc to be zero. Now the length of every path from $(1, 0)$ to (n, T) in the TEN is its total travel time, which is also its TDSPP-PW objective, since no waiting at arcs in M is possible. Every path in the TEN corresponds to a feasible solution of the TDSPP-PW, and, by Proposition 10, the optimal TDSPP-PW solution has a corresponding path in the TEN. The result follows. □

3.6 Cases of TDSPP-LW Solvable in Polynomial Time

3.6.1 Waiting not allowed ($W = 0$) at a subset of nodes $M \subset N$

When $W = 0$, i.e., waiting at nodes in M is not allowed, the TDSPP-LW resembles TDSPP-PW with $\alpha > 1$, where waiting at nodes in M is discouraged. Indeed, the proof that this variant is solvable in polynomial time relies on showing that this variant of TDSPP-LW has a relaxation which is a variant of TDSPP-PW, and that solving this relaxation yields an optimal solution to the TDSPP-LW.

Theorem 8. *The variant of TDSPP-LW in which waiting is not allowed ($W = 0$) at a subset of nodes $M \subset N$ is solvable in polynomial time.*

Proof. Any feasible path for an instance of TDSPP-LW with $W = 0$ is a feasible path for the corresponding instance of TDSPP-PW with $\alpha > 1$ (i.e., with the same tally set M) having the same objective value. Hence, TDSPP-PW with $\alpha > 1$ is a relaxation of TDSPP-LW with $W = 0$. By Theorem 7, solving the variant of TDSPP-PW with $M \subset N$ with $\alpha > 1$ can be done in polynomial time and, by Proposition 10, yields an optimal TDSPP-

PW solution P^* that does not use any waiting arcs at nodes in M. In other words, for $i \in M$, we have that $\omega(P^*, i) = 0$, hence P^* is feasible for the TDSPP-LW with $W = 0$. Since P^* is an optimal solution to a relaxation of TDSPP-LW and is also feasible, it is also an optimal solution of TDSPP-LW, and has been found in polynomial time. □

3.6.2 Waiting constrained at $M \subset N$ with $M = N \setminus \{n\}$ and a positive limit on total waiting time ($W > 0$)

The proofs in this section only use that $n \notin M$, hence also apply to the case where $M = N \setminus \{1, n\}$. For the case where $M = N \setminus \{1\}$, apply the transformation used in the proof of Proposition 4 to return an instance with $M = N \setminus \{n\}$.

Definition 16. *A* non-trivial *travel subpath is a travel subpath that starts and ends at timed copies of different nodes. In our context, the only possible trivial travel subpaths are* $(1, 0)$ *and* (n, T).

Definition 17. *A timed path P that begins at timed node (i, t_i) and ends at timed node (j, t_j) can be* completed *by prepending a latest departure time path from 1 to i (arriving at node i at time t_i) and appending an earliest arrival time path from j to n (departing from node j at time t_j), and, if the resulting timed path is within the time horizon, adding timed node $(1, 0)$ at the beginning and timed node (n, T) at the end, to form a feasible timed path P' for the MTTP. If in addition, $\omega(P') \leq W$ then P' is feasible for the TDSPP-LW. If the resulting timed path has timed nodes outside the time horizon, then the timed path P cannot be extended to a feasible solution for the TDSPP-LW and is removed from consideration.*

We begin by introducing three progressively more specific properties which an optimal timed path P may have. We then show that if an optimal timed path exists, it is always possible to convert it to an optimal path with each of these properties. Importantly, we show that if an optimal path with the most specific property exists, it can be found in polynomial time.

Property 3. *All travel subpaths of P except possibly the last non-trivial travel subpath contain a breakpoint. If $\omega(P) < W$, then the last non-trivial travel subpath also contains a breakpoint.*

Property 4. *Each travel subpath of P consists of a latest departure time path to a timed node concatenated with an earliest arrival time path from the same timed node, where the timed node is a breakpoint if one exists in the travel subpath.*

Property 5. *Let R_1, R_2 and R_3 be the first, second-to-last (if it exists), and last non-trivial travel subpaths of P, respectively, then R_1 starts at timed node $(1, t_1)$ with $t_1 \in \{t_1^1, t_1^2, \ldots t_1^K\}$ where t_1^k is the latest departure time from node 1 for arrival at the k^{th} breakpoint (given an arbitrary ordering of the breakpoints), R_2 ends at timed node (j, t_j) with $t_j \in \{t_j^1, t_j^2, \ldots t_j^K\}$ where t_j^k is the earliest arrival time at node j when departing at the k^{th} breakpoint, and R_3 ends at timed node (n, t_n) with $t_n \in \{t_n^1, t_n^2, \ldots t_n^K\}$ where t_n^k is the earliest arrival time at node n when departing at the k^{th} breakpoint.*

If P contains only one non-trivial travel subpath, then $R_1 = R_3$ solves the MATP from timed node $(1, t_1)$ via a breakpoint to timed node (n, t_n), where the allowable sets for t_1 and t_n are constructed while also considering $(1, W)$ as a breakpoint. Moreover, if the breakpoint is the k^{th} breakpoint, then $t_1 = t_1^k$ and $t_n = t_n^k$. In other words, P is the completion of a timed path with only one timed node (i, t), which is a breakpoint.

In the case where P contains more than one non-trivial travel subpath, if $\omega(P) < W$, the subpath of P that starts with R_1 and ends with R_3 solves the MTTP from node 1 to node n with time horizon $[t_1, t_n]$. Otherwise, if $\omega(P) = W$, the subpath of P that starts with R_1 and ends with R_2 solves the MTTP from node 1 to node j with time horizon $[t_1, t_j]$, where, in case there are only two non-trivial subpaths, we set $R_2 = R_3$.

Lemma 9. *Suppose there exists an optimal timed path, P, such that Property 5 holds, then an optimal timed path can be found in polynomial time.*

Proof. If P contains only one non-trivial travel subpath, then P must be one of the K timed

paths found by completing the K timed path containing only one timed node which is a breakpoint. Selecting the resulting timed path that is feasible for the TDSPP-LW with the least total travel time must return an optimal solution since one of the solutions returns P. This can be done in $\mathcal{O}(2K(MATP))$ time. Thus in this case, an optimal solution can be found in polynomial time.

Otherwise, if P contains more than one non-trivial travel subpath, we split into the two cases $\omega(P) < W$ and $\omega(P) = W$. Let R_1, R_2 and R_3 be the first, second-to-last, and last non-trivial travel subpaths of P, respectively, then by Property 5, R_1 starts at timed node $(1, t_1)$ with $t_1 \in \{t_1^1, t_1^2, \dots t_1^K\}$ where t_1^k is the latest departure time from node 1 for arrival at the k^{th} breakpoint (given an arbitrary ordering of the breakpoints), R_2 ends at timed node (j, t_j) with $t_j \in \{t_j^1, t_j^2, \dots t_j^K\}$ where t_j^k is the earliest arrival time at node j when departing at the k^{th} breakpoint, and R_3 ends at timed node (n, t_n) with $t_n \in \{t_n^1, t_n^2, \dots t_n^K\}$ where t_n^k is the earliest arrival time at node n when departing at the k^{th} breakpoint.

If $\omega(P) < W$, the sets $\{t_1^1, t_1^2, \dots t_1^K\}$ and $\{t_n^1, t_n^2, \dots, t_n^K\}$ can be found in $\mathcal{O}(2K(MATP))$ time, where $MATP$ is the complexity of solving the MATP. Since the subpath of P that start with R_1 and ends with R_3 solves the MTTP from node 1 to node n with time horizon $[t_1, t_n]$, it must be one of the K^2 MTTP solutions found by solving a MTTP for each pair $[t_1^\alpha, t_n^\beta]$ where $\alpha, \beta \in \{1, 2, \dots, K\}$. Completing each of the K^2 MTTP solutions and selecting the completed timed path that is feasible for the TDSPP-LW with the least total travel time must return an optimal solution. To see why, if each of the K^2 MTTP have unique solutions, one of the solutions contain a subpath of P which when completed returns P, otherwise, an alternative solution must have the same travel time as P as it solves the MTTP and has at most as much waiting time since it must arrive at node n no later than P. This can all be done in $\mathcal{O}(K^2(MTTP + MATP))$ time, where $MTTP$ is the complexity of solving the MTTP. Thus in this case, an optimal solution can be found in polynomial time.

If $\omega(P) = W$, suppose node j is known, then the sets $\{t_1^1, t_1^2, \dots t_1^K\}$ and

$\{t_j^1, t_j^2, \ldots, t_j^K\}$ can be found in $\mathcal{O}(2K(MATP))$ time, where $MATP$ is the complexity of solving the MATP. Since the subpath of P that start with R_1 and ends with R_2 solves the MTTP from node 1 to node j with time horizon $[t_1, t_j]$, it must be one of the K^2 MTTP solutions found by solving a MTTP for each pair $[t_1^\alpha, t_j^\beta]$ where $\alpha, \beta \in \{1, 2, \ldots, K\}$. Append waiting of length $W - \omega(P')$ at node j to each of the MTTP solutions found and complete the resulting timed paths. Selecting the completed timed path that is feasible for the TDSPP-LW with the least total travel time must return an optimal solution. To see why, if each of the K^2 MTTP have unique solutions, one of the solutions contain a subpath of P after appending waiting (since it is known that $\omega(P) = W$) and completing returns P, otherwise, an alternative solution must have the same travel time and tallied waiting time as the subpath of P as it solves the MTTP the sum of tallied waiting and travel time must be t_j, hence appending waiting also returns an optimal solution. This can be done in $\mathcal{O}(K^2(MTTP + MATP))$, where $MTTP$ is the complexity of solving the MTTP. Since node j is not known beforehand, this process needs to be repeated for each node $j \in \{2, 3, \ldots, n-1\}$ which introduces another factor of $\mathcal{O}(n)$ for a run-time of $\mathcal{O}(nK^2(MTTP + MATP))$. Thus in this case, an optimal solution can also be found in polynomial time. □

It now remains to show that such a P satisfying Property 5 exists, we shall do so by first showing that there exists P satisfying Property 3 and Property 4.

Lemma 10. *Any optimal timed path, P^*, can be converted into an optimal timed path, P, for which Property 3 holds.*

Proof. Let S_1 be the last non-trivial travel subpath of P^*, and suppose P^* contains at least one travel subpath $S_2 \neq S_1$ that does not contain a breakpoint. We claim that it is possible to apply Δ-shifting such that (i) tallied waiting does not increase, (ii) travel time does not change and (iii) after Δ-shifting, the travel subpath $S_2(\Delta)$ contains a breakpoint or merges with another travel subpath. It is clear that after such a Δ-shifting, the resulting timed

path is still feasible by (i), optimal by (ii) and contains one less travel subpath without a breakpoint by (iii), thus, by applying Δ-shifting repeatedly there exists an optimal timed path, P', such that all travel subpaths except possibly the last travel subpath contain a breakpoint.

Let P' be an optimal timed path such that all travel subpaths except possibly the last travel subpath contain a breakpoint. If $\omega(P') = W$, then set $P \leftarrow P'$ and we are done. Otherwise, if $\omega(P') < W$, consider Δ-shifting the last non-trivial travel subpath of P', S_3. The gradient of the function $\tau(S_3(\Delta))$ must be zero, otherwise there exists a Δ such that Δ-shifting S_3 reduces travel time while remaining feasible, since both positive and negatives values of Δ are allowed due to waiting on either side of S_3 as it does not contain a (n, T) which is a breakpoint. Now, perform Δ-shifting on S_3, selecting a value of Δ such that the resulting travel subpath must either (1) contain a breakpoint, in which case we are done, or (2) merges with another travel subpath which must contain a breakpoint since all other travel subpaths contain a breakpoint, in which case we are done or (3) the resulting optimal timed path P satisfies $\omega(P) = W$, in which case we are also done. Hence, in any case, the desired optimal timed path, P, exists.

We now prove the earlier claim. By Observation 2, there exists a range for Δ such that $\tau(S_k(\Delta))$ is an affine function of Δ in that range. Let m_k be the gradient of $\tau(S_k(\Delta))$. Note that if $m_2 = 0$, there exists a value of Δ such that Δ-shifting travel subpath S_2 by itself satisfies all of (i), (ii) and (iii), and thus the claim holds.

Assume now that $m_2 \neq 0$ and simultaneously shift S_1 by Δ and S_2 by $-\frac{m_1}{m_2}\Delta$. Note that the value $-\frac{m_1}{m_2}\Delta$ has been carefully chosen such that (ii) holds. Selecting $\Delta < 0$ and simultaneously shifting can only decrease the value of $\omega(P^*)$. To see why, consider the interval from the start of S_2 to the end of S_1, since travel time does not change the waiting in the interval does not change, waiting at node n may be introduced, but is not tallied, therefore, tallied waiting time can only decrease so (i) holds. If selecting a value of Δ such that either a travel subpath merges or contains a breakpoint, results in the travel

subpath $S_2(-\frac{m_1}{m_2}\Delta)$ containing a breakpoint or merging with an existing travel subpath that contains a breakpoint then (iii) holds and we are done. Otherwise, if the value of Δ results in $S_1(\Delta)$ containing a breakpoint or merging with an existing travel subpath, repeat the process, replacing S_1 with the merged travel subpath or simply updating the value of m_1 and the value of Δ until (iii) holds, which can occur, at most, a number of times equal to the sum of all breakpoints and the number of travel subpaths of P^*. In the worst case, the travel subpath containing a Δ-shifting of S_1 merges with the travel subpath containing a Δ-shifting of S_2 and we have $S_1 = S_2$ and (iii) holds. □

Lemma 11. *Any optimal timed path, P^*, for which Property 3 holds, can be converted into an optimal timed path, P, for which Property 3 and Property 4 hold.*

Proof. Suppose P^* contains at least one travel subpath S that does not consist of a latest departure time path to a timed node concatenated with an earliest arrival time path from the same timed node, where the timed node is selected to be a breakpoint if one exists in the travel subpath.

Select a breakpoint in S if one exists, otherwise, select any timed node in S. Replace S with S' in P, with S' being a latest departure time path from the first (untimed) node of S to the selected timed node concatenated with an earliest arrival time path from the same timed node to the last (untimed) node of S. The resulting timed path P' must satisfy Property 3, since S' contains a breakpoint if S contains a breakpoint. By construction, it must be that $\tau(S') < \tau(S)$, however, P' must not be feasible otherwise that would contradict the optimality of P^*, so it must be the case that $\omega(P') > W \geq \omega(P)$. Moreover, the decrease in travel time must be equal to the increase in waiting time, and since waiting at node n, which is not tallied, can only increase via this procedure, it must be the case that $\tau(P) - \tau(P') \geq \omega(P') - \omega(P)$.

Let R be the last non-trivial travel subpath of P' and apply Δ-shifting to R with $\Delta < 0$. Select Δ with $|\Delta|$ as small as possible such that either (i) the resulting timed path, P'', satisfies $\omega(P'') = W$ or (ii) $R(\Delta)$ contains a breakpoint or (iii) $R(\Delta)$ merges with an

existing travel subpath. In cases (ii) and (iii), repeat the procedure until case (i) holds, which can occur, at most, a number of times equal to the sum of all breakpoints and the number of travel subpaths of P'. Note that in each iteration, the increase in travel time must be equal to the decrease in waiting time, and since waiting at node n, which is not tallied, can only increase via this procedure, it must be the case that $\tau(P'') - \tau(P') \leq \omega(P') - \omega(P'')$. Since in the last iteration, P'' is feasible and $\omega(P'') \leq \omega(P') + \tau(P') - \tau(P'') \leq \omega(P) + \tau(P) - \tau(P'')$, we have $\tau(P'') \leq \tau(P) + \omega(P) - W \leq \tau(P)$, hence by setting $P \leftarrow P''$, we return an optimal timed path such that Property 4 hold. Property 3 holds since every travel subpath except the last non-trivial travel subpath contains a breakpoint and $\omega(P) = W$. □

Lemma 12. *Any optimal timed path, P^*, for which Property 3 and Property 4 hold, can be converted into an optimal timed path, P, for which Property 5 holds.*

Proof. Let S_1 and S_3 be the first and last non-trivial travel subpaths of P^* respectively, and suppose S_1 starts at timed node $(1, t_1)$, S_3 ends at timed node (n, t_n). Note that due to Property 3 and Property 4, it must be the case that $t_1 \in \{t_1^1, t_1^2, \ldots t_1^K\}$ where t_1^k is the latest departure time from node 1 for arrival at the k^{th} breakpoint (given an arbitrary ordering of the breakpoints) and $t_n \in \{t_n^1, t_n^2, \ldots t_n^K\}$ where t_n^k is the earliest arrival time at node n when departing at the k^{th} breakpoint.

Select an optimal solution to the MTTP from node 1 to node n with time horizon $[t_1, t_n]$ such that the solution satisfies Property 3 and Property 4. (The algorithm of He et al. (2019) for solving MTTP produces such a solution.) It must be the case that for any travel subpath of the solution that ends at a node (j, t_j), we have that $t_j \in \{t_j^1, t_j^2, \ldots t_j^K\}$, where t_j^k is the earliest arrival time at node j when departing at the k^{th} breakpoint. Complete the solution to the MTTP to obtain a timed path P', which must have $\tau(P') \leq \tau(P)$, since the subpath of P starting with S_1 and ending with S_3 is feasible for the MTTP.

Let R_1 and R_3 be the first and last non-trivial travel subpaths of P' respectively, and suppose R_1 starts at timed node $(1, t_1')$ and R_3 ends at timed node (n, t_n').

If $\omega(P') < W$ then we are done since the subpath of P' that starts with R_1 and ends

with R_3 solves the MTTP from node 1 to node n with time horizon $[t'_1, t'_n]$ since it solves the MTTP from node 1 to node n with larger time horizon $[t_1, t_n] \supseteq [t'_1, t'_n]$. If P' contains only one non-trivial travel subpath R, then since R is a feasible timed path for the MATP from node 1 to node n starting at t'_1 is must also be optimal for that MATP.

Otherwise if $\omega(P') \geq W$, apply Δ-shifting to R_3 with $\Delta \leq 0$. Select Δ with $|\Delta|$ as small as possible such that either (i) the resulting timed path, P'', satisfies $\omega(P'') = W$ or (ii) $R_3(\Delta)$ contains a breakpoint or (iii) $R_3(\Delta)$ merges with an existing travel subpath. In cases (ii) and (iii), repeat the procedure until case (i) holds, which can occur, at most, a number of times equal to the sum of all breakpoints and the number of travel subpaths of P'. Note that in each iteration, the increase in travel time must be equal to the decrease in waiting time, and since waiting at node n which is not tallied can only increase via this procedure, it must be the case that $\tau(P'') - \tau(P') \leq \omega(P') - \omega(P'')$. Since in the last iteration, P'' is feasible and $\omega(P'') \leq \omega(P') + \tau(P') - \tau(P'') \leq \omega(P) + \tau(P) - \tau(P'')$ so $\tau(P'') \leq \tau(P) + \omega(P) - W \leq \tau(P)$, hence by setting $P \leftarrow P''$, we return an optimal timed path for the TDSPP-LW. To see why Property 5 holds for P, note that any subpath of P' that starts at a timed node (i, t_i) and ends at a timed node (j, t_j) solves the MTTP from node i to node j with time horizon $[t_i, t_j]$, otherwise replacing that subpath with the MTTP solution provides a timed path with better objective value than P' which is a contradiction. Thus, if R_2 is the second-to-last travel subpath of P and ends at timed node (j, t_j), then the subpath of P (and also of P') that starts with R_1 and ends with R_2 solves an MTTP from node 1 to node j with time horizon $[t_1, t_j]$. If P' contains only one non-trivial travel subpath R that starts at timed node $(1, W)$ (since this is the only node where tallied waiting can occur and $\omega(P) = W$) and ends at timed node (n, t) then R must solve an MTTP from node 1 to node n with time horizon $[W, t]$ and since R is a feasible timed path for the MATP from node 1 to node n starting at W it must also be optimal for that MATP. □

Theorem 9. *The variant of TDSPP-LW in which waiting is constrained at intermediate nodes $M = N \setminus \{n\}$ and the limit on total waiting is positive ($W > 0$) is solvable in*

polynomial time.

Proof. If an optimal timed path exists, consecutively applying Lemma 10, Lemma 11, and Lemma 12 guarantees the existence of an optimal timed path, P, that satisfies Property 5.Lemma 9 specifies how an optimal solution can be found in polynomial time given the existence of such a timed path, P. □

We have recently been made aware of an alternative proof of Theorem 9, independently derived, provided in Omer and Poss (2019a).

3.7 Summary

We summarize the known complexity results for variants of TDSPP-PW and TDSPP-LW in Table 3.4 and Table 3.5. Note that PT stands for polynomial time and NPH stands for NP-Hard. The results in bold are the additions to Table 3.2 and Table 3.3 derived in Section 3.2-Section 3.3. Equivalence of the $N \setminus M = \{n\}$ and $N \setminus M = \{1\}$ cases in the third row follow from Corollary 2.

Table 3.4: Complexity results for variants of TDSPP-PW.

	$0 < \alpha \leq 1$	$\alpha > 1$
$M = N$	PT (Proposition 5)	**NPH (Theorem 2)**
$N \setminus M = \{1, n\}$	**PT (Theorem 6)**	PT (Proposition 6)
$N \setminus M = \{n\}$ or $\{1\}$	**PT (Theorem 6)**	PT (Proposition 7)
$M \subset N$	**PT (Theorem 6)**	**PT (Theorem 7)**

Table 3.5: Complexity results for variants of TDSPP-LW.

	$W = 0$	$W > 0$
$M = N$	**NPH (Theorem 3)**	**NPH (Theorem 3)**
$N \setminus M = \{1, n\}$	PT (MDP)	**PT (Theorem 9)**
$N \setminus M = \{n\}$ or $\{1\}$	PT (MATP)	**PT (Theorem 9)**
$M \subset N$	**PT (Theorem 8)**	**NPH (Theorem 4)**

3.8 Final Remarks

In this chapter, we have determined the complexity of some variants of time-dependent shortest path problems. Below, we provide our perspective on the practical value of these results.

The type of time-dependent shortest path problem we have studied arises in situations where there is a trade-off between waiting and traveling. This may occur, for example, in the transport of perishable goods, when product decay happens faster when traveling, rather than being stored at a hub. To minimize decay from product origin to destination, given time-dependent travel times, it may be advantageous to wait at intermediate locations. However, a limit on total waiting may be imposed. Another example occurs when planning a sightseeing itinerary. Tourists prefer exploring sights over traveling, so minimizing total travel time is a primary objective. The minimum time allotted for exploring a sight can be included in the (time-dependent) travel times between sights, and a constraint on waiting at less desirable sights can be imposed to allow for more time at the more desirable sights.

Another potential application lies in the case where the network is linear, i.e., when the route is known and only the travel time needs to be evaluated, which occurs frequently as a subproblem in vehicle routing problems (Chabrier 2006, Liberatore et al. 2011). Being able to solve TDSPP-LW and TDSPP-PW introduces greater flexibility in the parent problem and allows more difficult variants to be solved such as those in (Dabia et al. 2013, Behnke et al. 2021).

Note also that the "travel time" in this chapter is not restricted to modeling just travel time, but can also include other time-dependent properties, such as service time, processing time, and mandatory waiting time (that is not tallied).

Although the time-complexity of several polynomial-time variants may be discouraging at face value, the factors relating to n and K stem from having to solve many subproblems (MTTP and MATP) that are very similar. It is possible that exploiting this similarity com-

putationally may result in practically viable algorithms. Exploring such ideas is left for future research.

CHAPTER 4

THE SERVICE NETWORK DESIGN PROBLEM WITH HUB CAPACITY

4.1 Introduction

Before the turn of the century, it was not unusual for package delivery to take several weeks to arrive. Nowadays, the expectation for delivery times has risen dramatically, with an increasing number of retailers offering same-day or even two-hour delivery within metropolitan areas. In a recent survey (Windstream Enterprise Retail 2020), 32% of retailers surveyed had already implemented same-day delivery, while an additional 23% had plans to implement same-day delivery within the next 2 years. On the customer side, 66% said they would choose a store that offers same-day delivery over a store that does not offer it. Such retailers typically rely on courier companies and other third-party services such as Uber for their delivery services. The quality of the services offered is dictated by physical distance and cost-efficiency. Efforts to reduce physical distance may include using store inventory to fulfill online orders or leasing alternative storage space within cities.

In this paper, we tackle the problem of finding cost-efficient solutions to enable same-day, intra-city delivery. An assortment of packages, each with different origins and destinations are to be routed through a road network within the span of several hours. Hubs in which packages can be consolidated or separated are present within the city. The key to finding a cost-optimal solution lies in identifying the hubs in which packages should be consolidated and the timing of those consolidations. This comes with unique challenges due to the short time horizon for which deliveries must be scheduled and the limited space for loading and unloading of vehicles in an urban environment. The service network design problem has traditionally been used to model the routing of packages and consolidations through a graph that captures the underlying topology of the road network. However, the

underlying road network and time horizon for the service network design problem is usually large, meaning that the granularity of time is typically represented in days to quarter days. Hubs are constructed with the relative capacity required in mind, meaning that loading and unloading capacity is not usually a concern. In addition, the remoteness of many hub locations means that there are opportunities to increase capacity if needed. However, hubs within a city are more restricted in size and ability to expand. Loading and unloading capacity needs to be modeled explicitly especially since loading and unloading times are significant with respect to the time horizon.

4.2 Literature Review

The original problem of routing multiple commodities through a network is the *Multi-Commodity Network Flow* (MCNF) (Ford Jr and Fulkerson 1958b) which extends the single-commodity network flow problem (more commonly known as the network flow problem). However, some of the underlying assumptions of the MCNF are unrealistic for modeling package routing for our application. One of these is the assumption that flows are present throughout the network concurrently, meaning the location of flows does not change with time. Dynamic flows, also called flows over time, overcome this assumption and were studied for single commodities in Ford Jr and Fulkerson (1958a) with a discrete-time framework and in Fleischer and Tardos (1998) with continuous time. Another assumption in the network flow setting is that flows of the same commodity may be split up among different paths. However, when the flows are viewed as packages, this is not always possible. MCNF serves as a starting point for many of the models that we shall discuss below since it shares key properties of handling commodities with different origin-destination pairs and utilizes consolidations to reduce costs. The uncapacitated variant of the MCNF, in which arcs had unlimited capacity, can be solved using an exact Lagrangian-based branch-and-bound scheme that can handle problems with up to 1000 arcs and 600 commodities (Holmberg and Hellstrand 1998). For the capacitated variant, which is con-

sidered to be significantly harder, Lagrangian-based approaches such as those in Crainic et al. (2001) have also been used.

Multicommodity flows over time were introduced by Hall et al. (2007) as an extension of MCNF that models commodities that had release and due times and traveled through the network. They proved that the problem, even in the fractional flow case, was NP-hard (this contrasts with the MCNF, which is NP-hard only in the integer flow case).

Service network design problems are a class of tactical planning problems in freight transportation Magnanti and Wong (1984), Crainic and Rousseau (1986). They are used to inform decisions such as which transportation routes to provide, which transportation modes to provide, and when, while satisfying the requirements of the (carrier) service. While there is a significant portion of literature dedicated to making multimodal transportation choices (Kim et al. 1999, Alumur et al. 2012), we will instead focus on optimizing route choices and timings.

Several formulations for service network design problems appear in a review of service network design modeling and mathematical programming techniques in Crainic (2000) where the focus was addressing tactical-level decision making in "long-haul, intercity transportation". They distinguish between *frequency* and *dynamic* service network design. *Frequency* service network design addresses strategic-tactical level decisions about which services to offer, frequency of service over the planning horizon, traffic itineraries, terminal workloads, and policies. On the other hand, *dynamic* service network design addresses tactical-operational level decisions about *if* and *when* services depart. In the rest of the paper, when we refer to the service network design problem, we mean dynamic service network design rather than frequency service network design. In particular, the scheduled service network design problem.

Scheduled service network design is often modeled with the aid of a time-expanded network. Unfortunately, this approach leads to much large integer programs compared to the static case, and until recently, only heuristics were available to solve moderately sized

problems. The first exact approach to solve large-scale *Service Network Design Problem* (SNDP) was presented in Boland et al. (2017), where they applied an iterative approach that solved a sequence of simpler lower-bound relaxations on a time-expanded network while guaranteeing convergence to the original problem. The method was then improved upon in a subsequent paper (Marshall et al. 2020). This was not the first time iterative schemes have been proposed to solve problems on time-expanded networks that are too large to solve directly. Iterative refinement has been used in Wang and Regan (2002, 2009), Dash et al. (2012), Vu et al. (2020) to tackle the Time-Dependent Traveling Salesman Problem with Time Windows. It has also been used more recently in He et al. (2019) to solve Time-Dependent Shortest Path Problem and in Hewitt (2019) extends the approach of Boland et al. (2017) to the Continuous Time Load Plan Design Problem, while Scherr et al. (2020) applies it to the Service Network Design Problem with Mixed Autonomous Fleets.

Iterative refinement starts with a partially time-expanded network made up of only a small number of time points or time-buckets. These are then used to construct a lower-bound formulation for their original problem. By solving the lower-bound formulation, information is then gained that can be used to improve the partially time-expanded network. Where these approaches differ is the role the lower-bound formulation plays in solving the final problem and the way that refinement is made. Nonetheless, the ability to reduce the problem instance by several orders of magnitude is crucial in solving large-scale problems. This is only possible because even small partially time-expanded networks can contain the necessary information to solve the problem. In many regions of the time-expanded network, the aggregate information is enough to inform whether decisions should be made there.

Service Network Design Problem with Hub (Loading and Unloading) Capacity (SNDP-HC) was first considered in Wu et al. (2020) for the problem of same-day delivery faced by SF Express. In this scenario, same-day delivery services such as “12-18 service”, “14-20 service” and “12-20 service” are provided in several of China’s major cities. For example,

the "12-18 service" indicates that customers should present their package to couriers by 12:00 pm for their package to arrive at the specified destination by 6:00 pm. Packages are first consolidated at access points, then transported to the nearest hub. The service network design problem then covers the delivery of packages through the network of hubs so that each package arrives at the hub closest to its ultimate destination. The packages are then transported to access points from which they may be delivered individually. Therefore, the times of the service do not directly translate to time-windows of the service network design problem since time must be allocated to the "pre-processing" and "post-processing" steps of the delivery chain. The resulting time-windows for the service network operation are then around two to four hours. The authors argued that solving the integer program directly was not possible in a reasonable time and presented three heuristic approaches for the problem. Furthermore, they included an additional term in the objective function to maximize the number of commodities delivered, as it was not always possible to serve all commodities within the allotted time window. They presented results on real instances of 17, 31, 32 nodes and greater than 95% commodity density. While their solution approach is indeed fast, it does not provide any optimality guarantee. In this paper, we push the limits of the exact approach for this problem and show that although we are unable to solve instances of the same size, we can solve 15 node instances exactly within one day.

4.3 Problem Formulation

Let $D = (N, A)$ be a network with node set N and directed arc set A. The nodes in N represent hubs. Associated with each hub $i \in N$ is a loading time $t_i^l \in \mathbb{N}_{\geq 0}$, an unloading time $t_i^u \in \mathbb{N}_{\geq 0}$, a loading capacity $c_i^l \in \mathbb{N}_{\geq 0}$, and an unloading capacity $c_i^u \in \mathbb{N}_{\geq 0}$. The arcs in A represent connections between hubs. Associated with each arc $a = (i, j) \in A$ is a travel time $\tau_{ij} \in \mathbb{N}_{\geq 0}$, a fixed cost $f_{ij} \in \mathbb{R}_{\geq 0}$ representing the cost of sending a single vehicle along the arc and a per-unit cost $c_{ij} \in \mathbb{R}_{\geq 0}$ representing the cost per unit of commodity of sending a commodity along the arc. For the city logistics setting, the per-unit

cost is not as important, however, we set the per-unit cost to be a small number for purposes that will become apparent in Subsection 4.7.4. Let K denote a set of commodities, each of which has a single origin $o_k \in N$, a single destination $d_k \in N$, and a quantity $q_k \in \mathbb{N}_{\geq 0}$ that must be routed along a single path (i.e. the commodity cannot be split into two separate commodity paths) from origin to destination. The commodity $k \in K$ becomes available at its origin at time $e_k \in \mathbb{N}_{\geq 0}$ and needs to reach its destination by time $l_k \in \mathbb{N}_{\geq 0}$. This is a hard constraint and late arrivals are not permitted in this model. All times can also be shifted so that the earliest e_k starts at time zero and T is used to indicate the latest l_k. Vehicles with capacity $Q \in \mathbb{N}_{\geq 0}$ are available to move commodities along the arcs. Assumption 4 is a common assumption in the literature on SNDP, this assumption makes sense either when the size of packages are small relative to the capacity of the vehicles used, or when package sizes are relatively equal, or when commodities can be split across multiple vehicles traveling the same leg.

Assumption 4. *No packing problem needs to be solved when consolidating commodities into vehicles. That is, if multiple commodities with combined quantity q are scheduled to be transported along arc $((i,t),(j,\bar{t}))$, then the number of vehicles required to transport the commodities is given by $\lceil \frac{q}{Q} \rceil$, rather than having to solve a packing problem based on the individual commodity quantities.*

Service Network Design Problem with Hub (Loading and Unloading) Capacity (SNDP-HC) seeks to determine paths for the commodities and the resources required to transport the commodities along these paths so as to minimize the total cost, while ensuring that the time constraints on the commodities are respected, the capacity constraints of the vehicles are respected, and the capacity constraints of the hubs are respected. The loading time at a hub specifies how long it takes to load commodities onto a vehicle. Every commodity that is loaded onto a vehicle must be at the hub before the loading of the vehicle starts. Similarly, every commodity that is unloaded from a vehicle is available at that hub after the unloading ends. The loading capacity c_i^l at hub $i \in N$ implies that at any time, no more

than c_i^l vehicles can be loaded simultaneously, and the unloading capacity c_i^u at hub $i \in N$ implies that at any time, no more than c_i^u vehicles can be unloaded simultaneously. Note that it is possible to simultaneously load and unload vehicles. The hub capacities represent the number of loading and unloading docks available at the hub. Associated with each hub are loading and unloading times, t_i^l and t_i^u. As an example, if a vehicle begins loading at time t, the dock becomes available at exactly time $t + t_i^l$.

Let $\mathcal{D} = (\mathcal{N}, \mathcal{A}, \mathcal{H})$ be a time-expanded network for the problem, with timed node set $\mathcal{N}$, timed travel arc set $\mathcal{A}$, and timed waiting arc set $\mathcal{H}$. The set of timed nodes $\mathcal{N}$ consists of node-time pairs (i, t) where $i \in N$, the set of times for node i we denote by $\mathcal{T}_i$. The union of $\mathcal{T}_i$ forms the discretization $\mathcal{T}$. The set of timed travel arcs $\mathcal{A}$ consists of timed node pairs $((i, t), (j, \bar{t}))$ where the timed nodes (i, t) and $(i, \bar{t})$ belong in $\mathcal{N}$, the underlying arc in the flat network (i, j) is in the arc set A, and the time between the timed nodes must be sufficient, i.e. $\bar{t} \geq t + t_i^l + \tau_{ij} + t_i^u$. For SNDP-HC we do not require equality since vehicles may be parked outside the loading and unloading docks while waiting for them to become available. The set of timed holding arcs $\mathcal{H}$, consists of consecutive timed nodes in $\mathcal{N}$, $((i, t), (i, \bar{t}))$, where $\bar{t} = \min\{s | (i, s) \in \mathcal{N}, s > t\}$. For simplicity, we assume that all time related data in the problem are integer, so that all timed copies in $\mathcal{N}$ occur at integer times. A complete time-expanded network $\bar{\mathcal{D}} = (\bar{\mathcal{N}}, \bar{\mathcal{A}}, \bar{\mathcal{H}})$ can then be created by having a timed copy of each node $i \in N$ at every integer point $\{0, 1, 2, \ldots, T\}$ and including all feasible timed arcs.

4.4 Complexity of the SNDP-HC

It is well known that SNDP is NP-Hard, thus SNDP-HC which is a generalization of SNDP (when loading and unloading capacities are equal to the number of commodities or loading and unloading times are zero) is also NP-Hard. However, another property of the SNDP is that for any given data it is easy (in P) to verify whether the problem is feasible. This does not translate directly for SNDP-HC, due to the existence of loading/unloading capacities

and times.

We first ask a simpler question: Given a SNDP-HC instance on a network $D = (N, A)$, where $N = \{1, 2\}$ and $A = (1, 2)$, and commodities $k \in K$ each with the same origin and destination $(o_k, d_k) = (1, 2)$ but different time windows (e_k, l_k), unit loading and unloading capacities $lc_1 = uc_2 = 1$, and unit vehicle capacity and commodity quantity, is it possible to determine feasibility in polynomial time?

An important observation is that only the greater of t_1^l and t_2^u matters. Assume t_1^l is greater. Then for any feasible loading schedule (here we mean a loading schedule such that a commodity is loaded by time $l_k - t_2^u - \tau_{12}$), if we project the end of the loading times at node 1 to the start of unloading at node 2 by adding the travel time, we will obtain a feasible unloading schedule (where no two vehicles are being unloaded at the same time). For the case where t_2^u is greater, the projection will be from the start of unloading times at node 2 to the end of loading at node 1 instead. Therefore, if t_1^l is greater, then we only need to consider how to load the commodities at node 1. Otherwise, if t_2^u is greater, we only need to consider how to unload the commodities at node 2.

With this observation, the problem is a scheduling problem with release and due times on one machine and equal processing time. The number of jobs is the number of commodities. Minimizing the number of late jobs to be scheduled on a single machine can be done in polynomial time if processing times are equal $(1|p_i = p, r_i| \sum U_i)$ (Garey et al. 1981). Minimizing the weighted number of late jobs to be scheduled on a single machine can be done in polynomial time if processing times are equal (Baptiste 1999). Two algorithms are presented, one for the non-preemptive problem $(1|p_i = p, r_i| \sum w_i U_i)$. The problem is NP-Hard if the processing times are not identical (Johnson and Garey 1979). Hence, the simple version of the problem is polynomial-time solvable. However, for a more complex network, it may be possible for feasibility to be NP-Hard.

If Assumption 4 does not hold, one way in which NP-hardness can arise is in terms of packing commodities for consolidation. Previously, in SNDP, the packing problem only

appeared when determining optimality but not in determining feasibility. This was because if costs were disregarded then each commodity could travel on its own vehicle. However, for SNDP-HC, even if costs are disregarded, there is a limit to the number of vehicles that can be utilized due to the loading and unloading capacity of terminals. To see this, consider the decision version of the bin-packing problem, with L bins of size Q and a list of K items with sizes $q_1, \ldots, q_K$ to pack, which asks whether all K items can be packed into L bins. We can reduce this to an instance of SNDP-HC with K commodities with the same origin o and destination d, the same release time $e = 0$ and deadline $l = L$, and commodity k having quantity q_k. Let the number of loading docks at o and the number of unloading docks at d be 1, let the loading time $t_o^l = 1$, travel time $\tau_{od} = 0$, and unloading time $t_d^u = 0$. It is easy to see that only L vehicles may travel from o to d due to the loading time, number of loading docks and the deadline. If we let the vehicle capacity be of size Q, then a feasible solution to the SNDP-HC must have packed all K items into L vehicles, which returns an equivalent feasible solution to the bin-packing problem. Therefore, determining the feasibility of SNDP-HC with packing requirements is NP-hard since there is a polynomial reduction from an NP-hard problem (bin-packing).

4.5 Integer Program for the SNDP

If loading and unloading need not be considered (i.e. $t_i^l = t_i^u = 0$), there are no downsides to having commodities arriving early and so it is sufficient to include only timed arcs that have $\bar{t} = t + \tau_{ij}$. Consider variables x_{ij}^{kt} representing whether commodity k travels along arc (i, j) beginning at time t (with no intermediate nodes) and variables y_{ij}^t representing the number of vehicles required along arc (i, j) departing at time t. The mathematical programming formulation below, given in Boland et al. (2017) solves the service network

design problem:

$$z(\mathcal{D}) = \min \left\{ \sum_{((i,t),(j,\bar{t}))\in\mathcal{A}} (f_{ij} y_{ij}^{t} + c_{ij} \sum_{k\in\mathcal{K}} q_k x_{ij}^{kt}) \right\} \tag{4.1}$$

subject to:

$$\sum_{((i,t),(j,\bar{t}))\in\mathcal{A}\cup\mathcal{H}} x_{ij}^{k\bar{t}} - \sum_{((j,\bar{t}),(i,t))\in\mathcal{A}\cup\mathcal{H}} x_{ji}^{kt} = \begin{cases} 1 & \text{if } (i,t) = (o_k, e_k), \\ -1 & \text{if } (i,t) = (d_k, l_k), \forall k \in \mathcal{K}, (i,t) \in \mathcal{N}, \\ 0 & \text{otherwise;} \end{cases} \tag{4.2}$$

$$\sum_{k\in\mathcal{K}} q_k x_{ij}^{kt} \leq Q y_{ij}^{t}, \qquad \forall((i,t),(j,t+\tau_{ij})) \in \mathcal{A}; \tag{4.3}$$

$$x_{ij}^{kt} \in \{0,1\}, \qquad \forall((i,t),(j,\bar{t})) \in \mathcal{A}\cup\mathcal{H}, k \in \mathcal{K}; \tag{4.4}$$

$$y_{ij}^{t} \in \mathbb{N}_{\geq 0}, \qquad \forall((i,t),(j,\bar{t})) \in \mathcal{A}. \tag{4.5}$$

The objective in (4.1) serves to minimize the sum of fixed and variable costs. (4.2) is the flow balance constraint for each timed node (i,t) which ensures that the sum of flows in via both travel timed arcs and waiting timed arcs are balanced with the sum flowing out. (4.3) is the vehicle capacity constraint for each travel timed arc $((i,t),(j,t+\tau_{ij}))$ which counts the number of vehicles required to transport the packages along this leg. Recall that there are no constraints describing packing requirements due to Assumption 4.

4.6 Integer Program for the SNDP-HC

Wu et al. (2020) provides an integer program for the SNDP-HC. To incorporate the loading and unloading constraints, the definition of x_{ij}^{kt} and y_{ij}^{t} needs to include another index $\bar{t}$. The binary variables $x_{ij}^{kt\bar{t}}$ now represent loading commodity k in node i at time t and com-

pletion of unloading in node j at time $\bar{t}$. The integer variables $y_{ij}^{t\bar{t}}$ represent the number of vehicles required for all commodities in node i that begins loading at time t, and finishes unloading in node j at time $\bar{t}$. Each arc now comprises of loading, travel, possible waiting and unloading. Critically, Assumption 4 is kept.

SNDP-HC can then be written as the following mathematical programming problem:

$$z(\mathcal{D}) = \min \left\{ \sum_{((i,t),(j,\bar{t}))\in\mathcal{A}} (f_{ij} y_{ij}^{t\bar{t}} + c_{ij} \sum_{k\in\mathcal{K}} q_k x_{ij}^{kt\bar{t}}) \right\} \tag{4.6}$$

subject to:

$$\sum_{((i,t),(j,\bar{t}))\in\mathcal{A}\cup\mathcal{H}} x_{ij}^{kt\bar{t}} - \sum_{((j,\bar{t}),(i,t))\in\mathcal{A}\cup\mathcal{H}} x_{ji}^{k\bar{t}t} = \begin{cases} 1 & \text{if } (i,t) = (o_k, e_k), \\ -1 & \text{if } (i,t) = (d_k, l_k), \quad \forall k \in \mathcal{K}, (i,t) \in \mathcal{N}, \\ 0 & \text{otherwise;} \end{cases} \tag{4.7}$$

$$\sum_{((i,s),(j,\bar{s}))\in\mathcal{A}, s\in[t,t+t_i^l)} y_{ij}^{s\bar{s}} \leq c_i^l, \qquad \forall (i,t) \in \mathcal{N}; \tag{4.8}$$

$$\sum_{((j,\bar{s}),(i,s))\in\mathcal{A}, s\in[t,t+t_i^u)} y_{ji}^{\bar{s}s} \leq c_i^u, \qquad \forall (i,t) \in \mathcal{N}; \tag{4.9}$$

$$\sum_{k\in\mathcal{K}} q_k x_{ij}^{kt\bar{t}} \leq Q y_{ij}^{t\bar{t}}, \qquad \forall ((i,t),(j,\bar{t})) \in \mathcal{A}; \tag{4.10}$$

$$x_{ij}^{kt\bar{t}} \in \{0,1\}, \qquad \forall ((i,t),(j,\bar{t})) \in \mathcal{A}\cup\mathcal{H}, k \in \mathcal{K}; \tag{4.11}$$

$$y_{ij}^{t\bar{t}} \in \mathbb{N}_{\geq 0}, \qquad \forall ((i,t),(j,\bar{t})) \in \mathcal{A}. \tag{4.12}$$

This formulation is largely identical to the formulation of SNDP given in Boland et al. (2017), with (4.6), (4.7), (4.10),(4.11), (4.12) the same as (4.1), (4.2), (4.4), (4.5) except with another index for x and y. (4.8) and (4.9) model the new loading and unloading requirement, which limits the number of vehicles actively loading and unloading at a terminal at a given time. This type of constraint is similar to those found in time-indexed

formulations for single-machine scheduling problems (Sousa and Wolsey 1992, van den Akker et al. 1996). By including another index, the size of the resulting integer program increases from $\mathcal{O}(|K||\mathcal{N}|)$ variables and $\mathcal{O}(|K||\mathcal{N}|)$ constraints to $\mathcal{O}(|K||\mathcal{N}|^2)$ variables and $\mathcal{O}(|K||\mathcal{N}| + |\mathcal{N}|^2)$ constraints. The additional factor of $\mathcal{N}$ dramatically increases the difficulty of solving this problem.

4.7 Methodology

For typical applications, the number of hubs in a city $|N|$ is relatively small, around 10 to 30. On the other hand, if we consider actions to be happening on the timescale of minutes, the length of the time horizon $|T|$ will need to be around 100 to 400. Taking $|N| = 10$ and $T = 100$ results in integer programs that have on the order of a hundred million integer variables and one million constraints. Clearly, this SNDP-HC formulation cannot be solved directly for instances of interest. Additionally, the complexity in both the row and column space makes naive implementations of row-generation, column-generation, or Lagrangian relaxations difficult to construct. There is also no obvious block-decomposition of the constraint matrix due to the double-indexing of time. We propose to work with the discretization $\mathcal{T}$ as reducing this also reduces $\mathcal{N}$.

4.7.1 Overview

We shall take the approach of carefully reducing the granularity of the discretization, similarly to Boland et al. (2017), using dynamic discretization discovery. This scheme allows us to work with far smaller time-expanded networks while retaining just enough information to solve the original problem. A high-level overview of dynamic discretization discovery in our current context is outlined below:

1. (Initialization) Start with a subset of timed nodes. Add select timed arcs to create a partially time-expanded network.

2. Solve an integer program (LB), which serves as a lower-bound formulation on the original problem, on the partially time-expanded network.

3. Check whether the solution of (LB) can be converted to a feasible solution for the original problem using a mixed-integer program (FEAS). If it can, then it must be an optimal solution and we are done.

4. Otherwise, the solution of (LB) must be infeasible for the original problem. Use a mixed-integer program (INFEAS) to detect where the infeasibilities occur and to inform how the partially time-expanded network needs to be modified to cut off the solution of (LB).

5. Return to step 2.

A sufficient condition for finite termination is that after each iteration, at least one additional timed node is added to the partially time-expanded network and once all timed nodes in the complete time-expanded network have been added, the timed arc set of the partially time-expanded network should be identical to that of the complete time-expanded network. Clearly, we do not wish to end up with the complete time-expanded network and as such, termination should ideally occur when only a small fraction of timed nodes is present in the partially time-expanded network. We spend the next subsections going over the details of each of the steps. However, we first make a small detour to motivate one of the key ideas of this approach.

4.7.2 Interval Thinking

For SNDP, the lower-bound formulation SNDP(LB) used in Boland et al. (2017) is identical to their formulation of SNDP except that the underlying network is replaced with the partially time-expanded network. They formally prove, using invariant properties (given below) maintained throughout the course of the algorithm, that SNDP(LB) is a relaxation of SNDP.

Property 6 (Boland et al. (2017)). *For all commodities $k \in K$, the nodes (o_k, e_k) and (d_k, l_k) are in $\mathcal{N}$.*

Property 7 (Boland et al. (2017)). *Every timed arc $((i,t),(j,\bar{t})) \in \mathcal{A}$ has $\bar{t} \leq t + \tau_{ij}$.*

Property 8 (Boland et al. (2017)). *For every arc $a = (i,j) \in A$ and for every node $(i,t) \in \mathcal{N}$, there is a timed arc of $((i,t),(j,\bar{t}))$ for some $\bar{t}$.*

Theorem 10 (Boland et al. (2017)). *Let $\mathcal{D}$ be a partially time-expanded network that satisfies Property 6,Property 7 and Property 8. Then SNDP(LB) is a relaxation of SNDP.*

Property 9 (Boland et al. (2017)). *If arc $((i,t),(j,\bar{t})) \in \mathcal{A}$, then there does not exist a node $(j,\bar{t}') \in \mathcal{N}$ with $\bar{t} < \bar{t}' \leq t + \tau_{ij}$.*

Theorem 11 (Boland et al. (2017)). *For a fixed $\mathcal{N}$, among the partially time-expanded networks $\mathcal{D}$ satisfying Property 7 and Property 8, the one that satisfies Property Property 9 induces an instance of SNDP(LB) with the largest optimal objective value.*

The core idea, which we also use in this paper, is as follows: to ensure that the partially time-expanded network can provide a lower-bound when used for SNDP(LB), it is sufficient that any feasible solution to SNDP can be converted to a feasible solution of the same or lower cost when solving SNDP(LB). This property is maintained by aggregating the functionality of timed nodes that are not present in the partially time-expanded network with timed nodes that are present. In particular, each timed node (i,t) in the partially time-expanded network represents the *interval* between it and the next timed copy of the node $(i,\bar{t})$ in the partially time-expanded network. For example, if $\{(0,0),(0,1),(0,2),(0,3)\}$ are timed nodes in the complete time-expanded network but only $\{(0,0),(0,3)\}$ are present in the partially time-expanded network then the timed node $(0,0)$ takes on the functionality of itself and both $(0,1)$ and $(0,2)$. This means that any arc that would depart at $(0,1)$ or $(0,2)$ has its tail node redirected to $(0,0)$. Similarly, arcs that arrive at $(0,1)$ or $(0,2)$ have

their head nodes redirected to $(0, 0)$. This guarantees that each arc in the complete time-expanded network "remains" in the partially time-expanded network. Also, any commodity path in the complete time-expanded network can be traced as a continuous commodity path in the partially time-expanded since the departure and arrival times of each leg (arc) are rounded down to the nearest timed node, preventing any discontinuities.

4.7.3 Initialization

Using the idea from the previous section, it suffices to start with only timed nodes for the earliest and latest times of each node. This discretization leads to a partially time-expanded network with the fewest nodes and arcs without changing (4.7). However, since all the activity has been aggregated to the earliest times for each node, many arcs will no longer be traveling strictly forward in time. We shall further explore the effects of different initializations in Section 4.9.

4.7.4 Creating a Lower-Bound Integer Program

To construct a valid lower-bound integer program for SNDP-HC, more needs to be done than for SNDP. It is not sufficient to simply rely on the changes in the partially time-expanded network, constraints in the SNDP-HC formulation also need to be adjusted to reflect the aggregate nature of both timed nodes and timed arcs in the partially time-expanded network. Due to Theorem 10, redirecting the timed arcs is sufficient for the flow balance and vehicle capacity constraints. It remains to treat the loading and unloading capacity constraints.

The loading and unloading capacity constraints in the lower-bound formulation should depend on the coarseness of the discretization. Previously, loading capacity constraints were present for each timed node and restricted the sum of the outgoing vehicle variables y in the subsequent interval of length equal to the loading time t_i^l to the loading capacity c_i^l. Timed nodes in the partially time-expanded network now represent events in the interval

from the current timed copy of the node to the next available timed copy of the node. If the length L of that interval is less than the loading time then t_i^l, c_i^l is an upper-bound on the loading capacity available for the node since no more than c_i^l vehicle loadings can occur in a time period of length L. Otherwise, the capacity should be no more than $c_i^l \lceil \frac{L}{t_i^l} \rceil$. These values are valid upper-bounds on the loading capacity and using them results in a valid relaxed constraint for the lower-bound formulation.

The loading and unloading constraints can be further strengthened by including additional variables as illustrated in the following example. Suppose we have $t_i^l = 5$ and $c_i^l = 1$, if the partially time-expanded network contains timed nodes $(i, 0)$, $(i, 2)$, $(i, 4)$, $(i, 6)$, then one lower-bound constraint for loading at timed node $(i, 0)$ is $\sum_j y_{ij}^0 \leq 1$, as the number of vehicles that begin loading in the interval $[0, 2)$ must be less than the capacity. Similarly, the constraint $\sum_j y_{ij}^0 + y_{ij}^2 \leq 1$ is also valid, since the number of vehicles that begin loading in the interval $[0, 4)$ must still be less than the capacity, which is a stronger constraint than when only one timed node was considered. However, the constraint $\sum_j y_{ij}^0 + y_{ij}^2 + y_{ij}^4 \leq 1$ is not valid since it is possible for one vehicle to begin loading at $(i, 0)$ and another to begin loading at $(i, 5)$, which would be represented in the partially time-expanded network as loading at $(i, 4)$. Formalizing this argument requires some additional notation presented below.

Given a partially time-expanded network $\mathcal{D} = (\mathcal{N}, \mathcal{A}, \mathcal{H})$, we introduce the operators $next(t; i)$ and $prev(t; i)$ on any node-time pair (i, t) to indicate the next and previous times of (i, t) in the network. Formally, $next(t; i) = \min_{\bar{t}}\{(i, \bar{t}) \in \mathcal{N} | \bar{t} > t\}$ and $prev(t; i) = \max_{\bar{t}}\{(i, \bar{t}) \in \mathcal{N} | \bar{t} \leq t\}$. Note that (i, t) need not be in $\mathcal{N}$. Then, to build the loading constraint for (i, t), define $\kappa_{i,t}^l = \lceil \frac{next(t;i)-t}{c_i^l} \rceil$, which indicates how many sets of loading events could occur in the interval represented by timed node (i, t).

The lower-bound integer program (SNDP-HC(LB)) is given by:

$$\underline{z}(\mathcal{D}) = \min \left\{ \sum_{((i,t),(j,\bar{t}))\in\mathcal{A}} (f_{ij} y_{ij}^{t\bar{t}} + c_{ij} \sum_{k\in\mathcal{K}} q_k x_{ij}^{kt\bar{t}}) \right\} \tag{4.13}$$

subject to:

$$\sum_{((i,t),(j,\bar{t}))\in\mathcal{A}\cup\mathcal{H}} x_{ij}^{kt\bar{t}} - \sum_{((j,\bar{t}),(i,t))\in\mathcal{A}\cup\mathcal{H}} x_{ji}^{k\bar{t}t} = \begin{cases} 1 & \text{if } (i,t) = (o_k, e_k), \\ -1 & \text{if } (i,t) = (d_k, l_k), \quad \forall k \in \mathcal{K}, (i,t) \in \mathcal{N}, \\ 0 & \text{otherwise;} \end{cases} \tag{4.14}$$

$$\sum_{((i,s),(j,\bar{s}))\in\mathcal{A}, next(s;i)\in[t,t+\kappa_{i,t}^l t_i^l)} y_{ij}^{s\bar{s}} \leq c_i^l \kappa_{i,t}^l, \qquad \forall (i,t) \in \mathcal{N}; \tag{4.15}$$

$$\sum_{((j,\bar{s}),(i,s))\in\mathcal{A}, next(s;i)\in[t,t+\kappa_{i,t}^u t_i^u)} y_{ji}^{\bar{s}s} \leq c_i^u \kappa_{i,t}^u, \qquad \forall (i,t) \in \mathcal{N}; \tag{4.16}$$

$$\sum_{k\in\mathcal{K}} q_k x_{ij}^{kt\bar{t}} \leq Q y_{ij}^{t\bar{t}}, \qquad \forall ((i,t),(j,\bar{t})) \in \mathcal{A}; \tag{4.17}$$

$$x_{ij}^{kt\bar{t}} \in \{0,1\}, \quad \forall ((i,t),(j,\bar{t})) \in \mathcal{A} \cup \mathcal{H}, k \in \mathcal{K}; \tag{4.18}$$

$$y_{ij}^{t\bar{t}} \in \mathbb{N}_{\geq 0}, \qquad \forall ((i,t),(j,\bar{t})) \in \mathcal{A}. \tag{4.19}$$

(4.13), (4.14), (4.17), (4.18), (4.19) are equivalent to (4.6), (4.7), (4.10), (4.11), (4.12) except for the differences in the underlying network. Additionally, c_{ij} is set to a small constant to prevent commodities being sent on cycles, due to the possibility of arcs traveling backwards in time in the partially time-expanded network. (4.8) and (4.9) have been replaced by (4.15) and (4.16), which scale the loading and unloading capacity based on the size of the interval represented by each timed node (i,t).

We provide a proof that SNDP-HC(LB) is a relaxation of SNDP-HC. To do so, it is sufficient to show that there is a corresponding solution $(\bar{x}, \bar{y})$ of lower cost in SNDP-HC(LB).

Theorem 12. *Let $\mathcal{D}$ be a partially time-expanded network that satisfies Property 6,Property 7 and Property 8. Then SNDP-HC(LB) is a relaxation of SNDP-HC.*

Proof. Let (x, y) be a feasible solution of SNDP-HC. Define $S_i^t = \{s : prev(s; i) = t\}$. We claim that mapping $\bar{x}_{ij}^{kt\bar{t}} = \sum_{s \in S_i^t} \sum_{\bar{s} \in S_j^{\bar{t}}} x_{ij}^{ks\bar{s}}$ and $\bar{y}_{ij}^{t\bar{t}} = \sum_{s \in S_i^t} \sum_{\bar{s} \in S_j^{\bar{t}}} y_{ij}^{s\bar{s}}$ produces a solution of lower cost in SNDP-HC(LB).

Firstly, this mapping respects the domain constraints for x and y. It is easy to see why the domain constraint of y is respected since the sum of non-negative integers is a non-negative integer. To see why the domain constraint of x is respected, note that for a given (i, j), at most one value of $x_{ij}^{ks\bar{s}}$ can be nonzero as commodity paths only visit each node at most once, therefore the sum is either zero or one.

This mapping preserves the cost of the objective function in (4.13) as every variable is part of exactly one of the summations of the mapped variables. Therefore we can write:

$$\begin{aligned} & \sum_{((i,t),(j,\bar{t})) \in \mathcal{A}} (f_{ij} \bar{y}_{ij}^{t\bar{t}} + c_{ij} \sum_{k \in \mathcal{K}} q_k \bar{x}_{ij}^{kt\bar{t}}) \\ = & \sum_{((i,t),(j,\bar{t})) \in \mathcal{A}_c} (f_{ij} y_{ij}^{t\bar{t}} + c_{ij} \sum_{k \in \mathcal{K}} q_k x_{ij}^{kt\bar{t}}) \end{aligned}$$

To see why (4.14) is satisfied, observe that the equation can be rewritten as:

$$
\begin{aligned}
&\sum_{((i,t),(j,\bar{t}))\in\mathcal{A}\cup\mathcal{H}} \bar{x}_{ij}^{kt\bar{t}} - \sum_{((j,\bar{t}),(i,t))\in\mathcal{A}\cup\mathcal{H}} \bar{x}_{ji}^{k\bar{t}t} \\
&= \sum_{((i,t),(j,\bar{t}))\in\mathcal{A}\cup\mathcal{H}} \sum_{s\in S_i^t} \sum_{\bar{s}\in S_j^{\bar{t}}} x_{ij}^{ks\bar{s}} - \sum_{((j,\bar{t}),(i,t))\in\mathcal{A}\cup\mathcal{H}} \sum_{\bar{s}\in S_j^{\bar{t}}} \sum_{s\in S_i^t} x_{ji}^{k\bar{s}s} \\
&= \sum_{s\in S_i^t} \left\{ \sum_{((i,t),(j,\bar{t}))\in\mathcal{A}\cup\mathcal{H}} \sum_{\bar{s}\in S_j^{\bar{t}}} x_{ij}^{ks\bar{s}} - \sum_{((j,\bar{t}),(i,t))\in\mathcal{A}\cup\mathcal{H}} \sum_{\bar{s}\in S_j^{\bar{t}}} x_{ji}^{k\bar{s}s} \right\} \\
&= \sum_{s\in S_i^t} \left\{ \sum_{((i,t),(j,\bar{s}))\in\mathcal{A}_c\cup\mathcal{H}_c} x_{ij}^{ks\bar{s}} - \sum_{((j,\bar{s}),(i,t))\in\mathcal{A}_c\cup\mathcal{H}_c} x_{ji}^{k\bar{s}s} \right\} \\
&= \sum_{s\in S_i^t} \mathbb{1}((i,s)=(o_k,e_k)) - \mathbb{1}((i,s)=(d_k,l_k)) \\
&= \begin{cases} 1 & \text{if } (i,t)=(o_k,e_k), \\ -1 & \text{if } (i,t)=(d_k,l_k), \quad \forall k\in\mathcal{K}, (i,t)\in\mathcal{N}, \\ 0 & \text{otherwise.} \end{cases}
\end{aligned}
$$

The last line follows from Property 6 since $(i,s)=(o_k,e_k)$ if and only if $i=o_k$ and $s=e_k$. Similarly, $(i,s)=(d_k,l_k)$ if and only if $i=d_k$ and $s=l_k$. To see why (4.17) is satisfied, observe that for all timed arcs ($((i,t),(j,\bar{t}))\in\mathcal{A}$) the equation can be rewritten as:

$$
\begin{aligned}
\sum_{k\in\mathcal{K}} q_k \bar{x}_{ij}^{kt\bar{t}} &= \sum_{k\in\mathcal{K}} q_k \sum_{s\in S_i^t} \sum_{\bar{s}\in S_j^{\bar{t}}} x_{ij}^{ks\bar{s}} \\
&= \sum_{s\in S_i^t} \sum_{\bar{s}\in S_j^{\bar{t}}} \sum_{k\in\mathcal{K}} q_k x_{ij}^{ks\bar{s}} \\
&\leq \sum_{s\in S_i^t} \sum_{\bar{s}\in S_j^{\bar{t}}} Q y_{ij}^{s\bar{s}} \\
&= Q y_{ij}^{t\bar{t}}.
\end{aligned}
$$

Finally, (4.15) can be written as:

$$
\begin{aligned}
&\sum_{((i,r),(j,\bar{r}))\in\mathcal{A},next(r;i)\in[t,t+\kappa^l_{i,t}t^l_i)} \bar{y}^{r\bar{r}}_{ij} \\
&= \sum_{((i,r),(j,\bar{r}))\in\mathcal{A},next(r;i)\in[t,t+\kappa^l_{i,t}t^l_i)} \sum_{s\in S^r_i}\sum_{\bar{s}\in S^{\bar{r}}_j} y^{s\bar{s}}_{ij} \\
&= \sum_{((i,s),(j,\bar{s}))\in\mathcal{A},s\in[t,t+t^l_i)} y^{s\bar{s}}_{ij} + \sum_{((i,s),(j,\bar{s}))\in\mathcal{A},s\in[t+t^l_i,t+2t^l_i)} y^{s\bar{s}}_{ij} + \dots \\
&+ \sum_{((i,s),(j,\bar{s}))\in\mathcal{A},s\in[t+(\kappa^l_{i,t}-1)t^l_i,t+\kappa^l_{i,t}t^l_i)} y^{s\bar{s}}_{ij} \\
&\leq c^l_i + c^l_i + \dots + c^l_i \\
&= c^l_i\kappa^l_{i,t}, \forall (i,t) \in \mathcal{N}.
\end{aligned}
$$

Where we have used $next(r;i) \in [a,b)$ implies that $S^r_i \subset [a,b)$. Similarly, (4.16) is also satisfied. Therefore, the mapping from (x,y) to $(\bar{x},\bar{y})$ does indeed produce a feasible solution of the same cost in SNDP-HC(LB), and therefore SNDP-HC(LB) is a relaxation of SNDP-HC.

To see why SNDP-HC(LB) produces a lower-bound, observe that it is possible that the vehicle capacity, loading capacity and unloading capacity constraints might not be tight. It may be possible to reduce the value of $\bar{y}$ while remaining feasible, reducing the objective value. □

4.7.5 Determining Feasibility

To determine the feasibility of the lower-bound solution, we solve a mixed-integer linear program SNDP-HC(OPT) that "counts" the number of infeasibilities in the solution. If SNDP-HC(OPT) returns an objective value of zero, then there are no infeasibilities, and the lower-bound solution can be converted to an optimal solution for the original problem. Oth-

erwise, SNDP-HC(OPT) will identify where infeasibilities lie. This information will then be used to determine which timed nodes should be added to "cut-off" this lower-bound solution from the lower-bound IP. Note that this corresponds to combining the mixed-integer programs FEAS and INFEAS in Subsection 4.7.1 into a single optimization problem.

SNDP-HC(LB) provides a lower-bound on the objective value. Furthermore, it also provides a solution that captures two important characteristics, the commodity path flows $\{P_k = (k^{(1)} = o_k, k^{(2)}, k^{(|P_k|)} = d_k) | k \in \mathcal{K}\}$ and the consolidation set $\mathcal{J} = \{v = (i, j, \bar{K}) | \bar{K} \subset K, (i, j) \in A\}$. Each consolidation v in the consolidation set is identified by an arc (i, j) and a set of commodities that travel together along that arc $\bar{K}$. The other characteristic of the solution, the timings for each arc, is discarded. The reason for not using the timings is because the timings have been "rounded-down", and therefore inaccurate for reconstructing a solution for the original problem. In fact, the primary goal of SNDP-HC(OPT) will be to figure out how to time the commodity paths and consolidations to return a feasible solution for the original problem. To allow SNDP-HC(OPT) to always have a feasible solution, we permit arcs to take on *short* travel times, given by the length of the arc in the partially time-expanded network, at a penalty to the objective function.

For each consolidation v, let $J_v = \{(k, j) | k \in v, v \text{ is the } j^{\text{th}} \text{ leg of } k\}$ be the set of commodities paired with their leg number. Let $\bar{\mathcal{J}} \subset \mathcal{J}$ be the set of consolidations for which J_v contains more than one commodity. Variables γ_j^k indicate the time of departure for the j^{th} leg of commodity k, while variables θ_j^k reflect the travel time used on that leg. θ_j^k takes values equal to a non-negative time spent waiting plus either the true travel time τ_j^k or the short travel time $\bar{\tau}_j^k$. The binary variables σ_j^k are used to track which of the cases for θ_j^k hold. If θ_j^k cannot be bounded below by the true travel time τ_j^k then σ takes the value of 1 which increases the value of the objective function, otherwise, it takes the value of 0 and does not contribute to the value of the objective function.

Below is the formulation to "correct" the lower-bound solution given in Boland et al. (2017) (with some minor modifications), we refer to this formulation as SNDP-TTF(CB)

(travel time feasibility, commodity-based). Note that this formulation does not consider loading or unloading capacity.

$$\min \sum_{k \in \mathcal{K}} \sum_{j=1}^{|P_k|-1} \sigma_j^k \tag{4.20}$$

subject to:

$$\theta_j^k \geq \tau_j^k (1 - \sigma_j^k) + \bar{\tau}_j^k \sigma_j^k \qquad \forall j = 1, \ldots, |P_k| - 1, \forall k \in \mathcal{K}, \tag{4.21}$$

$$\gamma_j^k + \theta_j^k \leq \gamma_{j+1}^k \qquad \forall j = 1, \ldots, |P_k| - 1, \forall k \in \mathcal{K}, \tag{4.22}$$

$$e_k \leq \gamma_{o_k}^k \qquad \forall k \in \mathcal{K} \tag{4.23}$$

$$\gamma_{|P_k|-1}^k + \theta_{|P_k|-1,d_k} \leq l_k \qquad \forall k \in \mathcal{K} \tag{4.24}$$

$$\gamma_{j_1}^{k_1} = \gamma_{j_2}^{k_2} \qquad \forall (k_1, j_1), (k_2, j_2) \in J_v, \forall v \in \bar{\mathcal{J}} \tag{4.25}$$

$$\theta_j^k \in \mathbb{R} \qquad \forall j = 1, \ldots, |P_k| - 1, \forall k \in \mathcal{K}, \tag{4.26}$$

$$\gamma_j^k \geq 0 \qquad \forall j = 1, \ldots, |P_k| - 1, \forall k \in \mathcal{K}, \tag{4.27}$$

$$\sigma_j^k \in \{0, 1\} \qquad \forall j = 1, \ldots, |P_k| - 1, \forall k \in \mathcal{K}, \tag{4.28}$$

The objective in (4.20) is to minimize the number of commodities that do not use the true travel time. (4.21) sets the travel time variable θ to be either the true travel time or the short travel time depending on variable σ. (4.22) ensures that the times between consecutive departure variables γ_j^k and γ_{j+1}^k are at least as long as the travel time variable θ. (4.23) and (4.24) enforce that the departure and arrival for commodity k occurs after the release and deadline times respectively. Finally, (4.25) is the linking constraint that connects commodities that have a common consolidation.

Consolidation-based formulation

Note that there is an equivalent consolidation-based formulation, where instead of indexing variables by commodity, variables are indexed by consolidations. Let γ^v denote the time

for which consolidation v begins loading and let τ^v and $\bar{\tau}^v$ denote the true travel time and short travel time respectively of consolidation v. Then the formulation SNDP-TTF(VB) (travel time feasibility, vehicle/consolidation based) can be written as:

$$\min \sum_{v \in \mathcal{J}} \sigma_v \tag{4.29}$$

subject to:

$$\theta_v \geq \tau_v(1 - \sigma_v) + \bar{\tau}_v \sigma_v \qquad \forall v \in \mathcal{J}, \tag{4.30}$$

$$\gamma_{v_1} + \theta_{v_1} \leq \gamma_{v_2} \qquad \forall v_2 \in \mathrm{suc}\,(v_1), \forall v_1 \in \mathcal{J}, \tag{4.31}$$

$$\theta_v \in \mathbb{R} \qquad \forall v \in \mathcal{J}, \tag{4.32}$$

$$\gamma_v \in [\eta_v, \lambda_v] \qquad \forall v \in \mathcal{J}, \tag{4.33}$$

$$\sigma_v \in \{0, 1\} \qquad \forall v \in \mathcal{J}. \tag{4.34}$$

The advantages of the consolidation-based formulation are the reduced number of variables and constraints. (4.20), (4.22), (4.21), (4.26), (4.28) are simply rewritten in terms of consolidations to produce (4.29), (4.31), (4.30), (4.32) and (4.34). The time-window constraints in (4.23) and (4.24) are captured by calculating time-windows $[\eta_v, \lambda_v]$ for each consolidation and merging with (4.27) to form (4.33). The early time for a consolidation η_v is the earliest time all commodities in the consolidation can arrive at the tail node of v, thus for each commodity in v, start from their release time and successively add the short travel times of the commodity legs until it reaches consolidation v, the largest of these times is then chosen to be η_v. A similar process is used to calculate λ_v which is the latest time all commodities in the consolidation must arrive at the tail node of v, except that the process starts from the commodity due times and proceeds backward through legs. Preprocessing must also be performed to construct $\mathrm{suc}\,(v)$ which is the set of consolidations that can only occur after consolidation v due to containing the same commodity but at a later leg. We

shall use this formulation as a base and include constraints and variables to model loading and unloading capacity. The resulting formulation is called SNDP-HC(OPT). As the formulation is quite long, we split the explanation into separate sections below.

Variables

We introduce two classes of variables to model the sequence of loading and unloading, $l^t_{v_1v_2}$ to model loading consolidation v_2 after consolidation v_1 finishes loading at time t and $u^t_{v_1v_2}$ to model unloading consolidation v_2 after consolidation v_1 finishes unloading at time t. Note that for $l^t_{v_1v_2}$ only the variables for which $t \in [\eta_{v_1} + t^l_i, \lambda_{v_1} + t^l_i] \cap [\eta_{v_2}, \lambda_{v_2}] := \mathbb{T}^l_{v_1v_2}$, need to be considered, similarly, for $u^t_{v_1v_2}$ only the variables for which $t \in [\eta_{v_1} + \bar{\tau}_{v_1}, \lambda_{v_1} + \bar{\tau}_{v_1}] \cap [\eta_{v_2} + \bar{\tau}_{v_2} - t^u_i, \lambda_{v_2} + \bar{\tau}_{v_2} - t^u_i] := \mathbb{T}^u_{v_1v_2}$ need to be considered. The time windows are chosen with $\bar{\tau}$ on both sides since using τ may result in empty time windows leading to infeasibility of the MILP. There are additionally two more classes of variables to model whether loading and unloading capacity can be respected, w^l_i which counts the additional loading capacity needed for node i and w^u_i which counts the additional unloading capacity needed for node i.

Objective Function

The variables w^l_i and w^u_i are added to the objective function in (4.29), with weights μ^l_i and μ^u_i. The objective function for the MILP is given below:

$$z_O = \min \sum_{v \in \mathcal{J}} \sigma_v + \sum_{i \in N} (\mu^l_i w^l_i + \mu^u_i w^u_i) \tag{4.35}$$

The weights μ^l_i, μ^u_i on the variables w^l_i and w^u_i should reflect the number of timed nodes that will be added in the case that w is nonzero. The values will become clear when we introduce our scheme for adding timed nodes.

Changes to travel time constraints

(4.30), (4.31), (4.32), (4.33), (4.34) remain the same. However, the travel time parameters τ,$\bar{\tau}$ and the variable θ now incorporate loading and unloading times. For example, τ_j^k (which will be written as τ_v) represents loading, travel and unloading time required for a consolidation (that contains commodity k) traversing arc $(k^{(j)}, k^{(j+1)})$, however, this does not include any time spent waiting for an unloading dock to be available. The time spent waiting for an unloading dock to be available is implicitly modeled by the sequential unloading variables.

Loading and unloading constraints

To model the sequence of loading and unloading operations at a hub i, consolidations need to be grouped by origin and destination terminal. The notation $\mathcal{J}$ is extended to $\mathcal{J}_i^l$ to indicate consolidations originating from terminal i and $\mathcal{J}_i^u$ to indicate consolidations ending at terminal i. The null consolidation $v_{i\emptyset}$ is added to each of $\mathcal{J}_i^l$ and $\mathcal{J}_i^u$, to indicate the empty dock state and is given a time-window of $[\eta_{v_{i\emptyset}}, \lambda_{v_{i\emptyset}}] = [0, T]$. This is used to calculate time-windows $\mathbb{T}^l_{v_1 v_2}$ for loading from an empty consolidation (empty loading dock) or loading to an empty consolidation.

The loading constraints can then be written as:

$$\sum_{v \in J_i^l} l^0_{v_{i\emptyset} v} = c_i^l + w_i^l \qquad \forall i \in N, \tag{4.36}$$

$$\sum_{v \in J_i^l} \sum_{t \in \mathbb{T}^l_{v v_{i\emptyset}}} l^t_{v v_{i\emptyset}} = \sum_{v \in J_i^l} \sum_{t \in \mathbb{T}^l_{v_{i\emptyset} v}} l^{t+1}_{v_{i\emptyset} v} \qquad \forall t \in [0, T-1], \forall i \in N, \tag{4.37}$$

$$\sum_{v_1 \in J_i^l} \sum_{t \in \mathbb{T}^l_{v_1 v_2}} l^t_{v_1 v_2} = \sum_{v_1 \in J_i^l} \sum_{t + t_i^l \in \mathbb{T}^l_{v_2 v_1}} l^{t+t_i^l}_{v_2 v_1} \quad \forall t \in [\eta_{v_2}, \lambda_{v_2}], \forall v_2 \in J_i^l, v_2 \neq v_{i\emptyset}, \forall i \in N, \tag{4.38}$$

$$\sum_{v_1 \in J_i^l} \sum_{t \in \mathbb{T}^l_{v_1 v_2}} l^t_{v_1 v_2} = 1 \qquad \forall v_2 \in J_i^l, v_2 \neq v_{i\emptyset}, \forall i \in N, \tag{4.39}$$

$$l^t_{v_1 v_2} \in \{0, 1\}, l^t_{v_1, v_1} = 0 \qquad \forall t \in \mathbb{T}^l_{v_1 v_2}, \forall v_1, v_2 \in J_i^l, \forall i \in N, \tag{4.40}$$

$$l^t_{v_{i\emptyset} v_{i\emptyset}} \in \mathbb{Z}^+ \qquad \forall t \in [0, T], \forall i \in N, \tag{4.41}$$

$$w_i^l \in \mathbb{Z}^+ \qquad \forall i \in N, \tag{4.42}$$

(4.36) initializes the capacity of loading at each hub i to $c_i^l + w_i^l$. This capacity is then conserved via flow balance constraints (4.37) and (4.38), which checks that if loading for a consolidation starts at t that it ends at $t + t_i^l$ or $t + 1$ depending on whether the consolidation was empty. (4.39) enforces that loading begins exactly once for each consolidation. Note that the $l^t_{v_1 v_1}$ variables are set to zero.

Similarly, the unloading constraints can be written as:

$$\sum_{v \in J_i^u} u^T_{vv_{i\emptyset}} = c_i^u + w_i^u \qquad \forall i \in N, \tag{4.43}$$

$$\sum_{v \in J_i^u} \sum_{t \in \mathbb{T}^u_{v_{i\emptyset}v}} u^t_{v_{i\emptyset}v} = \sum_{v \in J_i^u} \sum_{t-1 \in \mathbb{T}^u_{vv_{i\emptyset}}} u^{t-1}_{vv_{i\emptyset}} \qquad \forall t \in [1,T], \forall i \in N, \tag{4.44}$$

$$\sum_{v_2 \in J_i^u} \sum_{t \in \mathbb{T}^u_{v_1v_2}} u^t_{v_1v_2} = \sum_{v_2 \in J_i^u} \sum_{t-t_i^u \in \mathbb{T}^u_{v_2v_1}} u^{t-t_i^u}_{v_2v_1} \quad \forall t \in [\eta_{v_1}, \lambda_{v_1}], \forall v_1 \in J_i^u, v_1 \neq v_{i\emptyset}, \forall i \in N, \tag{4.45}$$

$$\sum_{v_2 \in J_i^u} \sum_{t \in \mathbb{T}^u_{v_1v_2}} u^t_{v_1v_2} = 1 \qquad \forall v_1 \in J_i^u, v_1 \neq v_{i\emptyset}, \forall i \in N, \tag{4.46}$$

$$u^t_{v_1v_2} \in \{0,1\}, u^t_{v_1,v_1} = 0 \qquad \forall t \in \mathbb{T}^u_{v_1v_2}, \forall v_1, v_2 \in J_i^u, \forall i \in N, \tag{4.47}$$

$$u^t_{v_{i\emptyset}v_{i\emptyset}} \in \mathbb{Z}^+ \qquad \forall t \in [0,T], \forall i \in N, \tag{4.48}$$

$$w_i^u \in \mathbb{Z}^+ \qquad \forall i \in N, \tag{4.49}$$

Linking constraints

Finally, there needs to be linking constraints between the original variables θ, γ and the new variables l and u. (4.50) states that loading variables $l^t_{v_1v_2}$ should be synced with the departure variables γ_{v_2}. (4.51) states that unloading variable $u^t_{v_1v_2}$ should be set only after the earliest time of arrival, $\gamma_{v_1} + \theta_{v_1}$, which is necessary because of the possibility of waiting before unloading. Finally, (4.52) prevents consolidations from loading before it has finished unloading.

$$\gamma_{v_2} = \sum_{v_1 \in J_i^l} \sum_{t \in \mathbb{T}^l_{v_1 v_2}} t l^t_{v_1 v_2} \qquad \forall v_2 \in J_i^l, \forall i \in N \tag{4.50}$$

$$\gamma_{v_1} + \theta_{v_1} \leq \sum_{v_2 \in J_i^u} \sum_{t \in \mathbb{T}^u_{v_1 v_2}} t u^t_{v_1 v_2} \qquad \forall v_1 \in J_i^u, \forall i \in N \tag{4.51}$$

$$\gamma_{v_3} \geq \sum_{v_2 \in J_i^u} \sum_{t \in \mathbb{T}^u_{v_1 v_2}} t u^t_{v_1 v_2} \qquad \forall v_1 \in \mathcal{J}, \forall v_3 \in \mathrm{suc}\,(v_1). \tag{4.52}$$

(4.35) through (4.52) form the complete formulation of SNDP-HC(OPT).

4.7.6 Adding new time points

When the lower-bound solution (solution to SNDP-HC(LB)) exists but cannot be converted to a feasible solution (of SNDP-HC), additional timed nodes need to be added to the partially time-expanded network to refine the solution. Infeasibility of the lower-bound solution can manifest in two different ways, in either travel time infeasibility, or loading/unloading capacity infeasibility, determined by the values of the variables returned by SNDP-HC(OPT). If the value of σ_v is nonzero, then the travel time of consolidation v needs to be corrected for SNDP-HC(LB) to cut off the previous lower-bound solution. This is done by observing the arc that is used by consolidation v and adding a time point such that the arc is now of the correct length. For example, if the timed arc used for consolidation v was $((i, t_1), (j, t_2))$, and the true travel time of arc (i, j) was t_3, then the new timed node that we add is $(j, t_1 + t_3)$.

Otherwise, if the value of w_i^l or w_i^u (in SNDP-HC(OPT)) is nonzero, the loading or unloading capacity of node i is insufficient for the current set of consolidations. The new timed node should be a timed copy of node i, however, the timing is not explicit since the w variables do not correspond to times. A viable strategy is to look at the timeline of arrivals/departures at node i in the lower-bound solution. For each timed node (i, t), check whether the *true* loading/unloading capacity constraints in (4.8) and (4.9) hold. If there are

violations of the true constraints at timed node (i, t), then it may be useful to add the timed node $(i, t+t_i^l)$ as it will add the constraint to SNDP-HC(LB). For example, suppose we have a loading time of 8 and a loading capacity of 4 at node i. If the lower-bound solution has timed nodes (i, t), $(i, t + 4)$, $(i, t + 7)$, $(i, t + 15)$ with outbound consolidations numbering 3, 0, 2, 1 respectively, it satisfies the lower-bound loading capacity constraint at (i, t) since $3 + 0 \leq 4$ (here $\kappa_{i,t}^l = 1$). However, since it does not satisfy the true loading capacity constraint at (i, t), as $3 + 0 + 2 > 4$, adding the timed node $(i, t + 8)$ would "cut-off" that solution. Note that if all the arcs in the lower-bound solution are of the correct length and all timed nodes satisfy the true capacity constraints, then the solution is immediately feasible and optimal.

Since we are adding one timed node for each violation of true constraints, it is reasonable to set μ_i^l and μ_i^u to be equal to the number of violations of true loading and unloading constraints. For nodes that do not have violations, we fix the corresponding w variables to be zero, since the MILP is feasible with this additional restriction and if those w variables are non-zero, it will be unclear what timed nodes to add to the partially time-expanded network.

4.7.7 Complete algorithm and proof

Lemma 13. *In each iteration, if the algorithm does not terminate, at least one new timed node is added to the partially time-expanded network.*

Proof. Since the algorithm does not terminate, we have $z > 0$. Therefore, at least one of the variables σ_v or z_i^u or z_i^l is nonzero. If $\sigma_v \neq 0$, the timed arc $((i, t), (j, \bar{t}))$ used for consolidation v must not have the correct length, i.e. $\bar{t} < t + \tau_{ij}$. Due to Property 9, $(j, t + \tau_{ij})$ must not be in the partially time-expanded network, so we can add $(j, t + \tau_{ij})$ to the partially time-expanded network. Otherwise, if $z_i^l \neq 0$ or $z_i^u \neq 0$, there must be violations of the actual constraints at node i, or else the variable would have been fixed to zero. Having a violation of the actual loading constraint at (i, t) must be the result of

input : Initial partially time-expanded network $\mathcal{D} = (\mathcal{N}, \mathcal{A}, \mathcal{H})$, arc parameters $(\tau_{i,j}, f_{i,j})$ for each $(i,j) \in A$, commodity parameters $(o_k, d_k, q_k, e_k, l_k)$ for each $k \in K$, node parameters $(c_i^l, c_i^u, t_i^l, t_i^u)$ for each $i \in N$, and vehicle capacity Q.

output: flow variables x and vehicle variables y which yield the minimum cost service network design that satisfies hub capacities.

while *true* **do**
  $(x, y, \underline{z}) \leftarrow SNDP - HC(LB)(\mathcal{D})$
  $v = process(x, y)$
  $(\sigma, \theta, l, u, w, z_O) \leftarrow SNDP - HC(OPT)(v)$
  if $z = 0$ **then**
    break
  else
    for $v \in \{v' | \sigma_{v'} = 1\}$ **do**
      $\mathcal{D} = correctTravel(\mathcal{D}, v)$
    end
    for $i \in \{i' | w_{i'}^l > 0\}$ **do**
      $\mathcal{D} = correctLoading(\mathcal{D}, i)$
    end
    for $i \in \{i' | w_{i'}^u > 0\}$ **do**
      $\mathcal{D} = correctUnloading(\mathcal{D}, i)$
    end
  end
end
return $(x, y, \underline{z}) = process(\sigma, \theta)$

Algorithm 4: *Dynamic Discretization Discovery* (DDD) Algorithm for SNDP-HC

not having the timed node $(i, t + t_i^l)$, since the corresponding constraint in the lower-bound model coincides when both (i,t) and $(i, t + t_i^l)$ are in the partially time-expanded network. Therefore, we can add $(i, t + t_i^l)$ to the partially time-expanded network. Similarly, if there is a violation of the actual unloading constraint at (i,t), we can add $(i, t+t_i^u)$ to the partially time-expanded network. □

Theorem 13. *Algorithm 4 converges after a finite number of iterations.*

Proof. In each iteration, if the algorithm does not terminate, by Lemma 13 at least one new timed node is added. Since there is a finite number of timed nodes to add before we reach the complete time-expanded network, the algorithm must terminate at or before then. □

4.7.8 Worst-case analysis of Algorithm 4

We construct an example showing that in the worst-case, all timed copies of a node in the lower-bound formulation that we proposed are required to even determine the feasibility of a SNDP-HC instance.

Example 1

Consider a complete network on the node set $N = \{o, 0, 1, 2, \ldots, n\}$. Let the commodity o-d pairs be $\{(o, i)|i = 0, 1, \ldots, n\}$ with release-deadline pairs all being $(0, n + 1)$. Let the loading capacity and loading time at node o both be one unit, and at all other nodes be infinite (or n) and zero respectively. Unloading capacity and unloading time are set to infinite and zero at all nodes. Assign arc costs as follows:

$$c_{ij} = \begin{cases} n - j, & \text{for } i = o, j < n, \\ 1, & \text{for } i = o, j = n, \\ 1, & \text{for } i \neq o, i < j, \\ n, & \text{otherwise.} \end{cases}$$

Note that this instance of SNDP-HC is infeasible since there are $n + 1$ commodities but only n vehicles may be loaded at node o since there is a loading time of 1 and a minimum travel time of 1. Furthermore, consolidation is not possible, to show this, fix n and perform induction on the number of commodities K. When $K = 1$, the single commodity must be sent directly at time $t = 0$ to meet the deadline. When K is increased (one more commodity needs to be routed), the new commodity must be loaded at the next available time slot at the origin (which will be at time $K - 1$) since it cannot be consolidated with any of the existing commodities, or it will not arrive by the deadline. However, when K increases to $n + 1$, the option of loading the commodity at the next available time slot is insufficient for the last commodity to arrive in time, hence there is no feasible schedule.

To show that all timed copies of node o are required to determine infeasibility we show that if any timed copies of node o are missing (except the first and last timed copies since those are required for the lower-bound formulation to work), then there exists a feasible solution to the lower-bound problem. Suppose there exists a timed copy of o, (o, t), that is not in the partially time-expanded network, and without loss of generality we may assume that it is the first such timed copy of o. Note that t must not be 0 otherwise SNDP-HC on the partially time-expanded network is trivially infeasible due to commodity with o-d pair $(o, 0)$ not being able to arrive by the deadline (this infeasibility is different from the infeasibility of the original SNDP-HC). Construct a feasible solution to the SNDP-HC on the partially time-expanded network as follows. For commodities with destination node number less than or equal to t, send them directly in increasing order of their destination node. For commodities with destination node number greater than t, consolidate them all into one vehicle along with the commodity with destination node t. This vehicle departs at time $t-1$ due to the absence of timed node (o, t) (and the way we have relaxed the loading constraint) and arrives at a timed copy of node t with time no later than $n = t-1+1+n-j$ (due to the Property 7). The commodities on that vehicle can then each be sent directly from node t to their respective destinations arriving no later than time $n + 1$ (whereas in the presence timed node (o, t) they will arrive at time $n + 2$).

Example 2

For the previous example, it can be argued that only timed nodes which are multiples of the loading time are required. To extend the example so that the loading time is not fixed to one, the idea is to offset the release times of the commodities, this can be done directly or (if the goal is to have identical release and deadlines) introduce 'feeder' nodes (new origin nodes that 'feed' into the old origin).

Here we describe the first method, which is to offset the release times of the commodities, note that some other parameters need also to be adjusted.

Consider a complete network on the node set $N = \{o, 0, 1, 2, \ldots, n\}$. Let the commodity origin-destination pairs be $\{(o, i) | i = 0, 1, \ldots, n\}$ with release-deadline pairs being $(0, n + 2)$ for all commodities except the commodity with origin-destination pair $(o, 1)$ which has release-deadline pair being $(1, n + 2)$. Let the loading capacity and loading time at node o both be two units, and at all other nodes be infinite (or n) and zero respectively. Unloading capacity and unloading time are set to infinite and zero at all nodes. Assign arc costs as follows:

$$f_{ij} = \begin{cases} n - j, & \text{for } i = o, j < n, \\ 1, & \text{for } i = o, j = n, \\ 1, & \text{for } i \neq o, i < j, \\ n, & \text{otherwise.} \end{cases}$$

As before, proceeding by induction implies that the first n commodities must be loaded in increasing order of destination node for a feasible solution to be possible. The staggered release time of commodity with o-d pair $(o, 1)$ forces only one vehicle to be loaded at time 0, since there must be a loading dock available at time 1 to accommodate that commodity. This leads to one new vehicle being loaded at every time unit. The last commodity cannot be loaded in time at the origin and cannot be consolidated in time with any of the other commodities and hence the instance of SNDP-HC is infeasible.

The proof that all timed copies of the origin node are required follows Example 1.

4.8 Handling Infeasibility

Infeasibility can also be tackled directly by introducing binary variables ν_k that model whether or not a commodity k is served. This manifests in the objective function with a coefficient of $-M$, which is the penalty cost of not serving a commodity. The resulting formulation then replaces SNDP-HC and similar changes are made to SNDP-HC(LB). Here, the benefits of using a consolidation-centric formulation for SNDP-HC(OPT) shines,

since all changes relating to not serving commodities are in the preprocessing step before creating SNDP-HC(OPT).

$$z(\mathcal{D}) = \min \left\{ \sum_{((i,t),(j,\bar{t}))\in\mathcal{A}} (f_{ij} y_{ij}^{t\bar{t}} + c_{ij} \sum_{k\in\mathcal{K}} q_k x_{ij}^{kt\bar{t}}) + \sum_{k\in\mathcal{K}} M(1-\nu_k) \right\}$$

subject to:

$$\sum_{((i,t),(j,\bar{t}))\in\mathcal{A}\cup\mathcal{H}} x_{ij}^{kt\bar{t}} - \sum_{((j,\bar{t}),(i,t))\in\mathcal{A}\cup\mathcal{H}} x_{ji}^{k\bar{t}t} = \begin{cases} \nu_k & \text{if } (i,t) = (o_k, e_k), \\ -\nu_k & \text{if } (i,t) = (d_k, l_k), \quad \forall k \in \mathcal{K}, (i,t) \in \mathcal{N}, \\ 0 & \text{otherwise;} \end{cases}$$

$$\sum_{((i,s),(j,\bar{s}))\in\mathcal{A}, s\in[t,t+t_i^l)} y_{ij}^{s\bar{s}} \le c_i^l, \qquad \forall (i,t) \in \mathcal{N};$$

$$\sum_{((j,\bar{s}),(i,s))\in\mathcal{A}, s\in[t,t+t_i^u)} y_{ji}^{\bar{s}s} \le c_i^u, \qquad \forall (i,t) \in \mathcal{N};$$

$$\sum_{k\in\mathcal{K}} q_k x_{ij}^{kt\bar{t}} \le Q y_{ij}^{t\bar{t}}, \qquad \forall ((i,t),(j,\bar{t})) \in \mathcal{A};$$

$$x_{ij}^{kt\bar{t}} \in \{0,1\}, \qquad \forall ((i,t),(j,\bar{t})) \in \mathcal{A}\cup\mathcal{H}, k \in \mathcal{K};$$

$$y_{ij}^{t\bar{t}} \in \mathbb{N}_{\ge 0}, \qquad \forall ((i,t),(j,\bar{t})) \in \mathcal{A};$$

$$\nu_k \in \{0,1\}, \qquad \forall k \in \mathcal{K}.$$

4.9 Computational Results

4.9.1 Instances

We perform our computational experiments on real-world data provided by SF Express for the city of Guangzhou and Beijing, these are the same instances that were used in Wu et al. (2020). The experiments were run on 64-bit Linux machine with E5-2650v3 Intel

Xeon processor @2.3GHz and 512 GB RAM. A summary of the instance characteristics is given in Table 4.1 and Table 4.2, which include the number of commodities $|K|$, the size of the planning period T, the number of nodes $|N|$, and an upper bound on the number of commodities that can be served $|K'|$. Summary statistics, where the mean is denoted by a bar and the standard deviation by a tilde, are given for the commodity sizes q_k, availability times e_k, deadlines l_k, direct travel times for each commodity τ_k, arc travel times τ_{ij}, arc costs c_{ij}, loading and unloading capacities c_i^l and c_i^u. All times are given in minutes. For all the instances, the loading and unloading times t_i^l and t_i^u were set to be 10 minutes, while the vehicle capacity Q was set to be 400 units.

There are several key differences between the Beijing and Guangzhou data. Firstly, Guangzhou has far fewer hubs, 17 hubs compared to 32 for Beijing. Secondly, it has lower capacity for both loading and unloading. Finally, while both contain commodities with different deadlines, only Guangzhou contains commodities with different availability times.

Table 4.1: Instance features (demand)

Instance	Demand										
	$\|K\|$	T	$\bar{q}_k$	$\tilde{q}_k$	$\bar{e}_k$	$\tilde{e}_k$	$\bar{l}_k$	$\tilde{l}_k$	$\bar{\tau}_k$	$\tilde{\tau}_k$	$\|K'\|$
Guangzhou	270	150	32	28	13:09	15	15:02	8	43	13	264
Beijing	986	135	34	32	13:15	0	15:12	15	44	13	980

Table 4.2: Instance features (network)

Instance	Network								
	$\|N\|$	$\bar{\tau}_{ij}$	$\tilde{\tau}_{ij}$	$\bar{c}_{ij}$	$\tilde{c}_{ij}$	$\bar{c}_i^u$	$\tilde{c}_i^u$	$\bar{c}_i^l$	$\tilde{c}_i^l$
Guangzhou	17	43	14	25	13	4	0	3	0
Beijing	32	44	13	26	12	5	2	4	3

We generate feasible instances of size $|N| = n$ from the data via the following procedure:

1. Randomly select n nodes from the data to form the network N.
2. Select all arcs $a = (i, j)$ from the data where $i \in N$ and $j \in N$.
3. Select all commodities from the data that have both their origin and destination in N.

4. Retain all the parameters from the original problem for the selected nodes, arcs, and commodities.

Since loading and unloading capacity is unchanged, hubs in the Beijing instances will have more available capacity than hubs in the Guangzhou instances.

4.9.2 Algorithm design choices

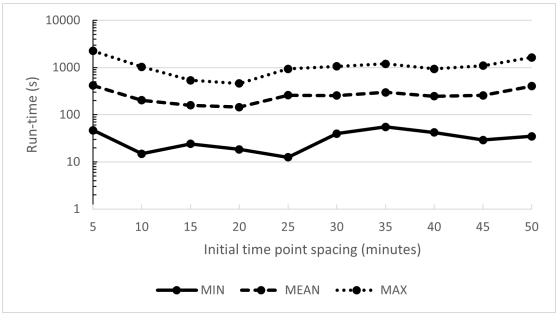
Several design decisions may have positive effects on the convergence of Algorithm 4. In this section, we perform a computational study to justify the design decisions we have made. Each design decision is discussed and compared across ten instances to evaluate the choice.

Selection of initial discretization - Coarseness

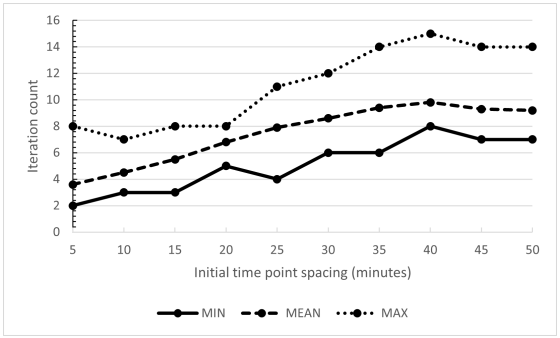
A crucial aspect of the dynamic discretization discovery framework is that we begin with a coarse discretization. However, this still leaves open how coarse the initial discretization should be. There is a trade-off between iteration count versus time per iteration since finer discretizations result in the algorithm requiring fewer iterations at the cost of more computation per iteration. Figure 4.1 and Figure 4.2 shows the effect of initial time point spacing on the run-time, iteration count and the number of time points added. The mean run-time appears to be at a minimum at the 10 minute mark for Beijing and the 20 minute mark for Guangzhou. The trend in the iteration count and number of time points added for coarser discretizations are as expected, with more iterations and a greater number of time points added.

Selection of initial discretization - Spacing

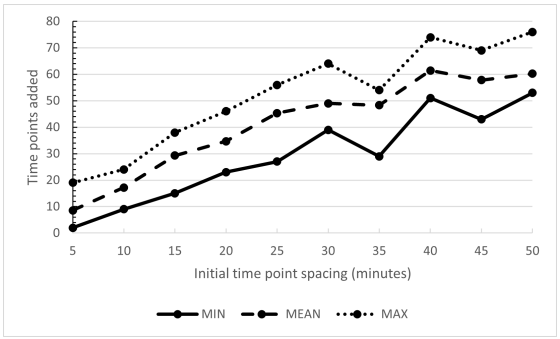
Other than having uniform discretizations across nodes and time, it is possible to have non-uniformity in both time and space. Several reasonable approaches would be to have a finer discretization near the edges of the time window as loading and unloading is more likely



(a) Run-time

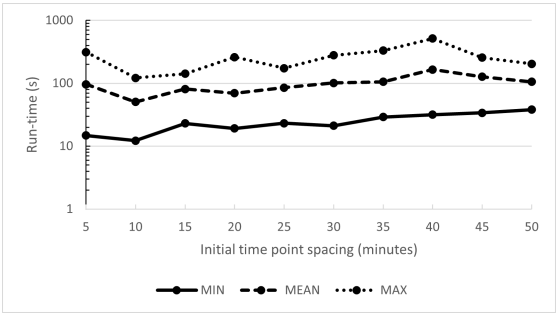


(b) Iteration count

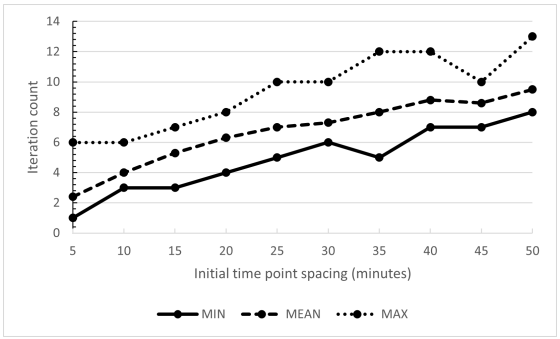


(c) Number of time points added

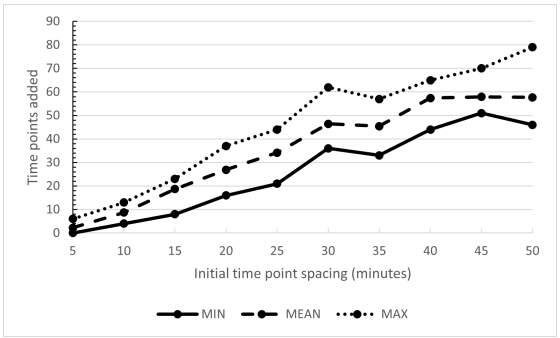
Figure 4.1: Effect of initial discretization (Beijing)



(a) Run-time



(b) Iteration count



(c) Number of time points added

Figure 4.2: Effect of initial discretization (Guangzhou)

to occur there. Note that since package release times are spread out in these instances, it is guaranteed that at the beginning of the time window that loading capacity is fully utilized since it is always beneficial to send packages out as early as possible even if they may need to wait for unloading. This is no longer true after an initial period since setting aside loading capacity for a package that has not yet arrived may be necessary. Using this knowledge, we experiment with a discretization scheme where nodes are spaced at time intervals t_i^l near the beginning and t_i^u near the end.

A different method of having non-uniform initial discretizations would be to discriminate across nodes. That is, identify likely hubs in the network where consolidation is likely to occur and assign a finer discretization in the middle of the planning horizon. In our experiments, such hubs were selected based on solving the SNDP-HC restricted to subgraphs of the network and looking at which hubs were used for consolidation in the optimal solution.

Separating travel time infeasibility and loading infeasibility

As noted in Subsection 4.7.1, there are two possible sources of infeasibilities in the lower-bound solutions. While Subsection 4.7.6 presents SNDP-HC(OPT) that deals with both sources simultaneously, it is also possible to resolve travel time infeasibilities first via solving SNDP-TTF(VB). If the lower-bound solution can be converted into a solution with no travel time infeasibilities, then the algorithm proceeds to solve SNDP-HC(OPT), otherwise, the algorithm proceeds directly to adding time points to remove the source of travel time infeasibility and skips over SNDP-HC(OPT) in this iteration. This may appear advantageous since SNDP-TTF(VB) is significantly easier to solve than SNDP-HC(OPT). Unfortunately, as we shall see in Figure 4.3 and Figure 4.4, since SNDP-HC(OPT) is a much smaller mathematical programming problem than SNDP-HC(LB), there are no gains to be made here.

Comparison of design decisions

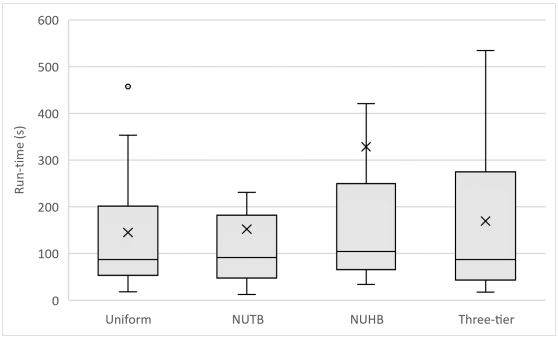
Figure 4.3 and Figure 4.4 evaluates the uniform two-tier (Uniform), non-uniform time-based (NUTB), non-uniform hub-based (NUHB), and uniform three-tier (Three-tier) approaches on 10 node subsets of Beijing and Guangzhou. Several discretizations were tried for the various strategies and the best one was picked to produce the figures. The three performance criteria pictured are the run-time, iteration count, and number of time points added. Surprisingly, the Uniform approaches work the best, owing mainly to the shorter time per iteration (compared to non-uniform approaches). Note that while there is a large variation in the run-time of different instances, with some instances taking up to 100 times longer than others, this is not unexpected since convergence depends on how well the discretization matches with the instance solution characteristics. For some instances, a lucky choice of discretization may discover the optimal solution after one iteration, whereas other instances may require exploring more areas of the time-expanded network to prune infeasible solutions.

4.9.3 Progression of lower-bound in algorithm

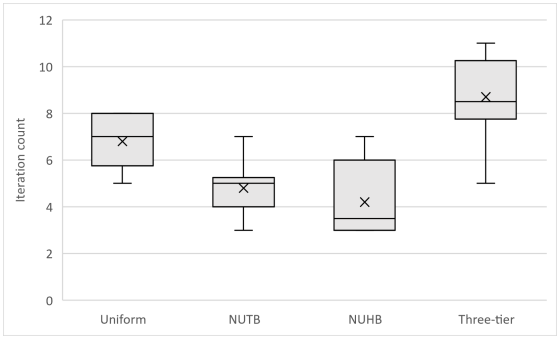
Using the two-tiered uniform approach in Subsection 4.9.2, Table 4.3 and Table 4.4 shows the progression of the lower-bound across ten different instances of size $n = 10$. Note that the Beijing instances were solved with a spacing of 20 minutes between time points, while the Guangzhou instances were solved with a spacing of 10 minutes between time points. This results in differences between the rates of convergence. Interestingly, the optimal objective value is sometimes found with an infeasible solution to the original problem, and requires another iteration, after adding additional time points, to find the optimal solution.

4.9.4 Comparison with full integer program

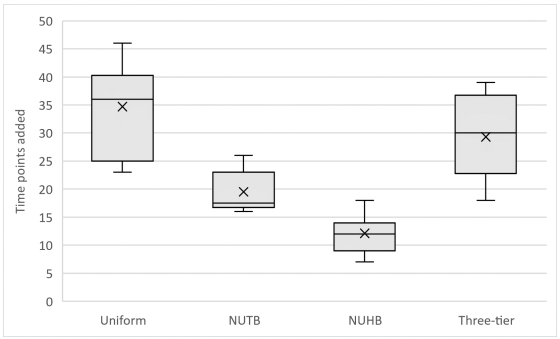
We compare the results of Algorithm 4 to the results of solving SNDP-HC directly, i.e. with time points spaced 1 minute apart. Table 4.5 and Table 4.6 contain the solve times and



(a) Run-time

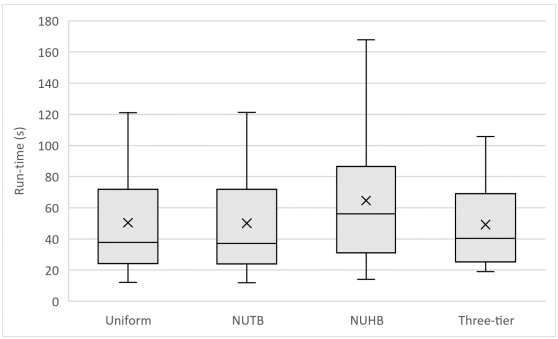


(b) Iteration count

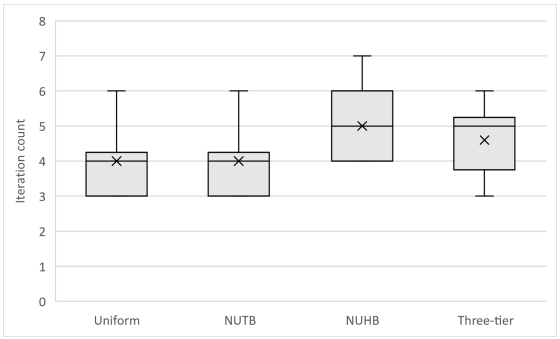


(c) Number of time points added

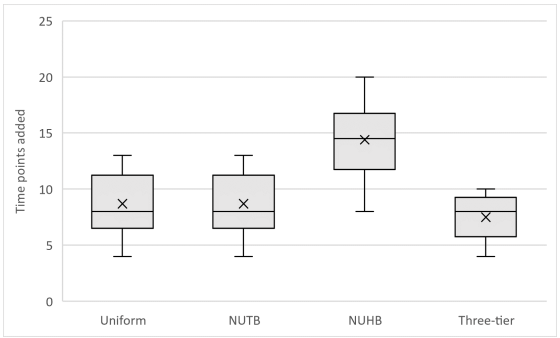
Figure 4.3: Effect of various design choices (Beijing)



(a) Run-time



(b) Iteration count



(c) Number of time points added

Figure 4.4: Effect of various design choices (Guangzhou)

Table 4.3: Progression of lower-bound (Beijing)

		Instance number										% of optimal
		1	2	3	4	5	6	7	8	9	10	
	1	752	858	1079	1687	989	1141	1187	869	1314	1330	69.6
	2	870	961	1228	1789	1022	1193	1402	972	1575	1629	78.6
	3	908	1109	1483	1902	1208	1363	1999	1136	1666	1850	90.9
Iteration	4	975	1109	1549	1991	1334	1510	2040	1170	1714	2008	95.7
Number	5	1005	1128	1575	2026	1401	1557	2040	1224	1743	2008	97.7
	6	1005	1139	1596	-	1448	1616	2058	1239	1785	-	98.9
	7	1005	1224	1608	-	1448	1616	-	1239	-	-	99.5
	8	1032	1232	1646	-	-	1616	-	-	-	-	100.0

Table 4.4: Progression of lower-bound (Guangzhou)

		Instance number										% of optimal
		1	2	3	4	5	6	7	8	9	10	
	1	1814	1798	2479	2020	1717	2192	2472	1746	2285	2205	92.7
	2	1986	1876	2502	2051	1727	2253	2655	1883	2427	2451	97.5
Iteration	3	2036	1964	2515	2058	1822	2336	2702	1908	2468	2472	99.6
Number	4	-	-	2524	2081	1822	2338	-	1916	2486	2474	99.9
	5	-	-	-	-	1838	2348	-	-	-	-	100.0
	6	-	-	-	-	-	2348	-	-	-	-	100.0

integer program sizes as well as the statistic for Algorithm 4 as a percentage of the statistic for SNDP-HC. While there are slight differences between the Beijing and Guangzhou instances (likely due to different spacing between time points), there are significant reductions of the integer program size in both cases, which translate to a speed-up of around 30 times on average. Unfortunately, the reduction in the integer program size does not directly correspond to a reduction in solve times, this is likely because the difficulty of the problem is still maintained in the reduced integer program.

Table 4.5: Aggregate Results - 10 node (Beijing)

	Algorithm 4	SNDP-HC	%
Average Solve Time (s)	145	3850	3.77
Variables (First Iteration)#	161742	13016210	1.24
Constraints (First Iteration)#	10352	250197	4.14
Variables (Last Iteration)#	237568	13016210	1.83
Constraints (Last Iteration)#	14325	250197	5.73

4.9.5 Solve times of larger instances

The solve times of 12 node instances for Beijing and Guangzhou, generated by the method in Subsection 4.9.1, are shown in Table 4.7. Just increasing the number of nodes in the problem by two significantly increases solve times by about a factor of 27 for the Beijing

Table 4.6: Aggregate Results - 10 node (Guangzhou)

	Algorithm 4	SNDP-HC	%
Average Solve Time (s)	50	1538	3.25
Variables (First Iteration)#	263860	16382148	1.61
Constraints (First Iteration)#	14869	292079	5.10
Variables (Last Iteration)#	286669	16382148	1.75
Constraints (Last Iteration)#	15906	292079	5.45

instances and 7 for the Guangzhou instances. Note that for the 7th instance of Guangzhou, the algorithm determined the instance is *infeasible* within 3 seconds, this result is not included in the average. Using the formulation in Section 4.8, the algorithm was able to find an optimal solution that served one less commodity in 258 seconds.

Table 4.7: Results - 12 node

Instance	Solve Times(s)										
	1	2	3	4	5	6	7	8	9	10	Average
Beijing	1550	1083	8148	271	594	2830	761	34863	1882	1169	5315
Guangzhou	167	385	267	1233	290	61	3*	84	228	416	348

Table 4.8 and Table 4.9 show the results of the largest instances we were able to solve. For both the Beijing and Guangzhou instances, we used a spacing of 10 minutes between timed nodes. The percentage reduction in the integer program size remains similar, even in the final iteration. The solve times for the individual instances are shown in Table 4.10. Four out of five of the 15 node Guangzhou instances generated were infeasible, but the formulation in Section 4.8 was unable to solve them in less than three hours.

Table 4.8: Aggregate Results - 15 node (Beijing)

	Algorithm 4	SNDP-HC	%
Solve Time(s)	37802	-	-
Variables (First Iteration)#	1504914	69475383	2.17
Constraints (First Iteration)#	51627	698788	7.39
Variables (Last Iteration)#	2105871	69475383	3.03
Constraints (Last Iteration)#	66775	698788	9.56

4.10 Conclusion

In this paper, we present a dynamic discretization discovery approach for SNDP-HC. Our approach successfully reduces the dimensions of the integer program by a factor of one

Table 4.9: Aggregate Results - 15 node (Guangzhou)

	Algorithm 4	SNDP-HC	%
Solve Time(s)	4540	-	-
Variables (First Iteration)#	1489059	94915657	1.57
Constraints (First Iteration)#	49046	842331	5.82
Variables (Last Iteration)#	1881255	94915657	1.98
Constraints (Last Iteration)#	58168	842331	6.90

Table 4.10: Results - 15 node

Instance	Solve Times(s)					
	1	2	3	4	5	Average
Beijing	40612	6946	73748	2199	65506	37802
Guangzhou	3475	2294	1337	7436	8160	4540

thousand, enabling instances of up to 10 nodes to be solved up to 30 times faster. While the algorithm alone may be insufficient for instances of greater than 15 nodes, it serves as a starting point for exact solution approaches. Decomposition approaches or heuristics can be used to solve SNDP-HC(LB) approximately or exactly. Other approaches to improve the algorithm may include using a heuristic to generate good solutions and selecting an initial discretization that contains the heuristic solution.

CHAPTER 5

CONCLUSION

5.1 Summary and Novelty of Dissertation

In Chapter 2, I present *Dynamic Discretization Discovery* (DDD) algorithms for two cases of time-dependent shortest path problems, one for the *Minimum Duration Problem* (MDP) and the other for the *Minimum Travel Time Problem* (MTTP). The *Dynamic Discretization Discovery* (DDD) algorithm for the MTTP relied on establishing the first polynomial-time algorithm for the MTTP, which is also presented in the chapter. All other existing algorithms have either exponential run-time or are inexact. The efficient solution procedures presented for these problems are especially important in path-planning under traffic or other operational constraints among other uses. One application of special interest is in the use of autonomous vehicles, where waiting may be discounted in the presence of free or cheap parking and no passenger is present. Consider the scenario in which an autonomous vehicle is used for commuting. When the primary user of the vehicle is at work, the autonomous vehicle is free to find a recharging station or perform other errands. This results in the vehicle being at a location other than the primary user's workplace. The vehicle should then solve a MTTP to return as this minimizes battery usage and reduces congestion for other vehicles.

In Chapter 3, I construct two broader classes of time-dependent shortest path problems, namely, those with penalties and limits on waiting. These classes of problems are each capable of generalizing the existing objectives for the time-dependent shortest path problem: arrival time, duration, and travel time. Furthermore, they provide additional modeling capability by allowing waiting in either the objective or the constraint, a parameter to control the amount of waiting, and the choice of only selecting a subset of the nodes for wait-

ing considerations. Certain selections of these parameters return the problems discussed in Chapter 2, while others have not been investigated before. I show which selections lead to polynomial-time solvable problems and present the corresponding polynomial-time algorithms, and present NP-hardness proofs for the remaining cases. This provides a significant contribution to existing literature and an additional modeling tool for practitioners interested in time-dependent shortest path problems with waiting such as applications in perishable good delivery.

In Chapter 4, I present a DDD algorithm for the *Service Network Design Problem with Hub (Loading and Unloading) Capacity* (SNDP-HC), which allows larger instances of SNDP-HC to be solved exactly for the first time. While the solution methodology is not capable of solving the largest instances, it provides an astonishing reduction in the size of integer programs solved, on the order of up to one thousand. This results in a speed-up of around 30 times for instances that could be solved previously, and can solve instances that could not be solved previously. The algorithm developed can be used as-is or to improve existing heuristics used to solve the SNDP-HC, an important problem in the routing of packages under a city logistics setting. In addition, the efficiency of DDD for this problem should motivate the exploration of DDD for other advanced classes of problems for which exact solution methods are not currently available.

5.2 Future Work

There are certainly many other individual problems that DDD can be applied to, I would like to conclude by giving a high-level overview of the potential research directions that my work leads into. While this dissertation focused on path and routing problems, it is important to recognize that DDD may apply to more general network problems.

Currently, the approach of DDD requires some level of tailoring to each specific application, which is no surprise as these problems differ greatly. However, it would be useful to formulate a general framework where a problem is given as a set of variables, parameters,

objectives, and constraints, which are decomposed into time-related and static, and see how precisely DDD can be worded in this case.

DDD can potentially be applied also in the spatial domain, especially when there is some level of homogeneity among groups of nodes at the spatial level and that the solution does not traverse large areas of space (so that they can remain aggregated). Note that spatial aggregation has been a common technique for heuristics and approximation algorithms, however, the power of DDD lies in selecting disaggregations to generate solutions within desired optimality bounds. DDD over time-expanded network can then be viewed as a special class of DDD over networks where "nodes" are aggregated only if they are timed copies of the same node. Of particular interest would be the use of DDD to identify nodes to disaggregate in neural networks to improve performance, which may allow the use of smaller initial networks that can grow to adapt to particular training sets.

Looking beyond time-indexed mathematical programming problems that are based upon time-expanded networks, similar concepts can be applied to stochastic optimization. Markov Decision Processes have similar structure to time-dependent shortest path problems except that timed arcs (now called actions) are dictated under a stochastic setting. Here, instead of a time discretization, it makes more sense to look at a state-space discretization instead, where states can be aggregated. The challenges would again be how a lower-bound (in expectation) formulation can be obtained and how states can be separated to improve the lower-bound.

Appendices

APPENDIX A

ARC TRAVEL TIME FUNCTIONS FOR THE EXAMPLE SHOWN IN Figure 2.4

Below are the functional forms of the arc travel time functions for the example given in Figure 2.4. These show the forward travel time on the arc as a function of the departure time at its tail node. Alongside each is shown the reverse travel time function, which gives the travel time on the arc as a function of the arrival time at its head node. The reverse travel time at each breakpoint of the reverse travel time function of each arc is shown in Table A.1.

$$c_{1,2}(t) = \begin{cases} 1.34 - 0.68t, & 0 \le t < 1 \\ 1.18 - 0.52t, & 1 \le t < 2 \\ 0.40 - 0.13t, & 2 \le t < 3 \\ -1.01 + 0.34t, & 3 \le t < 4 \\ -2.25 + 0.65t, & 4 \le t < 5 \end{cases} \qquad c_{1,2}^{rev}(t) = \begin{cases} 1.34 - 2.13t, & 1.34 \le t < 1.66 \\ 0.66 - 1.08t, & 1.66 \le t < 2.14 \\ 0.14 - 0.15t, & 2.14 \le t < 3.01 \\ 0.01 + 0.25t, & 3.01 \le t < 4.35 \\ 0.35 + 0.39t, & 4.35 \le t < 6.00 \end{cases}$$

$$c_{1,3}(t) = \begin{cases} 2.85 + 0.10t, & 0 \le t < 1 \\ 2.90 + 0.05t, & 1 \le t < 2 \\ 3.04 - 0.02t, & 2 \le t < 3 \\ 3.22 - 0.08t, & 3 \le t < 4 \\ 3.46 - 0.14t, & 4 \le t < 5 \end{cases} \qquad c_{1,3}^{rev}(t) = \begin{cases} 2.85 + 0.09t, & 2.85 \le t < 3.95 \\ 2.95 + 0.05t, & 3.95 \le t < 5.00 \\ 3.00 - 0.02t, & 5.00 \le t < 5.98 \\ 2.98 - 0.09t, & 5.98 \le t < 6.90 \\ 2.90 - 0.16t, & 6.90 \le t < 7.76 \end{cases}$$

$$c_{2,3}(t) = \begin{cases} 1.99 - 0.17t, & 0 \le t < 1 \\ 2.13 - 0.31t, & 1 \le t < 2 \\ 2.33 - 0.41t, & 2 \le t < 3 \\ 2.39 - 0.43t, & 3 \le t < 4 \\ 2.15 - 0.37t, & 4 \le t < 5 \end{cases} \qquad c_{2,3}^{rev}(t) = \begin{cases} 1.99 - 0.20t, & 1.99 \le t < 2.82 \\ 1.82 - 0.45t, & 2.82 \le t < 3.51 \\ 1.51 - 0.69t, & 3.51 \le t < 4.10 \\ 1.10 - 0.75t, & 4.10 \le t < 4.67 \\ 0.67 - 0.59t, & 4.67 \le t < 5.30 \end{cases}$$

$$c_{2,4}(t) = \begin{cases} 1.29 - 0.27t, & 0 \le t < 1 \\ 0.41 + 0.61t, & 1 \le t < 2 \\ -0.25 + 0.94t, & 2 \le t < 3 \\ 1.28 + 0.43t, & 3 \le t < 4 \\ 4.84 - 0.46t, & 4 \le t < 5 \end{cases} \qquad c_{2,4}^{rev}(t) = \begin{cases} 1.29 - 0.37t, & 1.29 \le t < 2.02 \\ 1.02 + 0.38t, & 2.02 \le t < 3.63 \\ 1.63 + 0.48t, & 3.63 \le t < 5.57 \\ 2.57 + 0.30t, & 5.57 \le t < 7.00 \\ 3.00 - 0.85t, & 7.00 \le t < 7.54 \end{cases}$$

$$c_{3,4}(t) = \begin{cases} 0.61 + 0.12t, & 0 \le t < 1 \\ 0.63 + 0.10t, & 1 \le t < 2 \\ 0.72 + 0.06t, & 2 \le t < 5 \end{cases} \qquad c_{3,4}^{rev}(t) = \begin{cases} 0.61 + 0.11t, & 0.61 \le t < 1.73 \\ 0.73 + 0.09t, & 1.73 \le t < 2.83 \\ 0.83 + 0.05t, & 2.83 \le t < 6.00 \end{cases}$$

Table A.1: Reverse Arc Travel Times at Each BP

BP Time	(1,2)	BP Time	(1,3)	BP Time	(2,3)	BP Time	(2,4)	BP Time	(3,4)
1.34	1.34	2.85	2.85	1.99	1.99	1.29	1.29	0.61	0.61
1.66	0.66	3.95	2.95	2.82	1.82	2.02	1.02	1.73	0.73
2.14	0.14	5.00	3.00	3.51	1.51	3.63	1.63	2.83	0.83
3.01	0.01	5.98	2.98	4.10	1.10	5.57	2.57	—	—
4.35	0.35	6.90	2.90	4.67	0.67	7.00	3.00	—	—
6.00	1.00	7.76	2.76	5.30	0.30	7.54	2.54	6.00	1.00

REFERENCES

S. A. Alumur, B. Y. Kara, and O. E. Karasan. Multimodal hub location and hub network design. *Omega*, 40(6):927–939, 2012.

P. Baptiste. Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine with equal processing times. *Journal of Scheduling*, 2(6):245–252, 1999.

J. Barcelo, Å. Hallefjord, E. Fernandez, and K. Jörnsten. Lagrangean relaxation and constraint generation procedures for capacitated plant location problems with single sourcing. *Operations-Research-Spektrum*, 12(2):79–88, 1990.

M. Behnke, T. Kirschstein, and C. Bierwirth. A column generation approach for an emission-oriented vehicle routing problem on a multigraph. *European Journal of Operational Research*, 288(3):794–809, 2021.

R. Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.

H. Belouadah, M. E. Posner, and C. N. Potts. Scheduling with release dates on a single machine to minimize total weighted completion time. *Discrete applied mathematics*, 36(3):213–231, 1992.

J. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische mathematik*, 4(1):238–252, 1962.

D. Bertsimas, A. Delarue, P. Jaillet, and S. Martin. Travel time estimation in the age of big data. *Operations Research*, 2019.

N. Boland, M. Hewitt, L. Marshall, and M. Savelsbergh. The continuous-time service network design problem. *Operations Research*, 65(5):1303–1321, 2017.

E. H. Bowman. The schedule-sequencing problem. *Operations Research*, 7(5):621–624, 1959.

I. Chabini. Discrete dynamic shortest path problems in transportation applications: Complexity and algorithms with optimal run time. *Transportation Research Record: Journal of the Transportation Research Board*, (1645):170–175, 1998.

A. Chabrier. Vehicle routing problem with elementary shortest path based column generation. *Computers & Operations Research*, 33(10):2972–2990, 2006.

Y. Chu and Q. Xia. Generating benders cuts for a general class of integer programming problems. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 127–141. Springer, 2004.

B. Coifman. Estimating travel times and vehicle trajectories on freeways using dual loop detectors. *Transportation Research Part A: Policy and Practice*, 36(4):351–364, 2002.

K. L. Cooke and E. Halsey. The shortest route through a network with time-dependent internodal transit times. *Journal of mathematical analysis and applications*, 14(3):493–498, 1966.

B. S. Cooper and R. V. Cowlagi. Path-planning with waiting in spatiotemporally-varying threat fields. *PloS one*, 13(8):e0202145, 2018.

G. Cote and M. A. Laughton. Large-scale mixed integer programming: Benders-type heuristics. *European Journal of Operational Research*, 16(3):327–333, 1984.

T. G. Crainic. Service network design in freight transportation. *European journal of operational research*, 122(2):272–288, 2000.

T. G. Crainic and J.-M. Rousseau. Multicommodity, multimode freight transportation: A general modeling and algorithmic framework for the service network design problem. *Transportation Research Part B: Methodological*, 20(3):225–242, 1986.

T. G. Crainic, A. Frangioni, and B. Gendron. Bundle-based relaxation methods for multicommodity capacitated fixed charge network design. *Discrete Applied Mathematics*, 112(1-3):73–99, 2001.

S. Dabia, S. Ropke, T. Van Woensel, and T. De Kok. Branch and price for the time-dependent vehicle routing problem with time windows. *Transportation science*, 47(3):380–396, 2013.

G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410, 1954.

G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations research*, 8 (1):101–111, 1960.

S. Dash, O. Günlük, A. Lodi, and A. Tramontani. A time bucket formulation for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 24(1):132–147, 2012.

B. C. Dean. Algorithms for minimum-cost paths in time-dependent networks with waiting policies. *Networks: An International Journal*, 44(1):41–46, 2004a.

B. C. Dean. Shortest paths in fifo time-dependent networks: Theory and algorithms. *Rapport technique, Massachusetts Institute of Technology*, 2004b.

U. Demiryurek, F. Banaei-Kashani, C. Shahabi, and A. Ranganathan. Online computation of fastest path in time-dependent spatial networks. In *International Symposium on Spatial and Temporal Databases*, pages 92–111. Springer, 2011.

M. Desrochers, J. Desrosiers, and M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations research*, 40(2):342–354, 1992.

B. Ding, J. X. Yu, and L. Qin. Finding time-dependent shortest paths over large graphs. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, pages 205–216. ACM, 2008.

D. Espinoza, R. Garcia, M. Goycoolea, G. L. Nemhauser, and M. W. Savelsbergh. Per-seat, on-demand air transportation part i: Problem description and an integer multicommodity flow model. *Transportation Science*, 42(3):263–278, 2008.

M. A. Figliozzi. The time dependent vehicle routing problem with time windows: Benchmark problems, an efficient solution algorithm, and solution characteristics. *Transportation Research Part E: Logistics and Transportation Review*, 48(3):616–636, 2012.

L. Fleischer and É. Tardos. Efficient continuous-time dynamic network flow algorithms. *Operations Research Letters*, 23(3-5):71–80, 1998.

L. R. Ford and D. R. Fulkerson. *Flows in networks*. Princeton University Press, 1962.

L. R. Ford Jr and D. R. Fulkerson. Constructing maximal dynamic flows from static flows. *Operations research*, 6(3):419–433, 1958a.

L. R. Ford Jr and D. R. Fulkerson. A suggested computation for maximal multi-commodity network flows. *Management Science*, 5(1):97–101, 1958b.

L. Foschini, J. Hershberger, and S. Suri. On the complexity of time-dependent shortest paths. *Algorithmica*, 68(4):1075–1097, 2014.

K. R. Fox, B. Gavish, and S. C. Graves. An n-constraint formulation of the (time-dependent) traveling salesman problem. *Operations Research*, 28(4):1018–1021, 1980.

A. Franceschetti, D. Honhon, G. Laporte, and T. Van Woensel. A shortest-path algorithm for the departure time and speed optimization problem. *Transportation Science*, 52(4):756–768, 2018.

M. Garey, D. S. Johnson, B. B. Simons, and R. E. Tarjan. Scheduling unit–time tasks with arbitrary release times and deadlines. *SIAM Journal on Computing*, 10(2):256–269, 1981.

F. Glover and M. Laguna. Tabu search. In *Handbook of combinatorial optimization*, pages 2093–2229. Springer, 1998.

C. Groër, B. Golden, and E. Wasil. A library of local search heuristics for the vehicle routing problem. *Mathematical Programming Computation*, 2(2):79–101, 2010.

V. M. Gunturi, K. Joseph, S. Shekhar, and K. M. Carley. Information lifetime aware analysis for dynamic social networks. Technical report, University of Minnesota, 2012.

S. K. Gupta and J. Kyparisis. Single machine scheduling research. *Omega*, 15(3):207–227, 1987.

A. Hall, S. Hippler, and M. Skutella. Multicommodity flows over time: Efficient algorithms and complexity. *Theoretical computer science*, 379(3):387–404, 2007.

C. A. Hane, C. Barnhart, E. L. Johnson, R. E. Marsten, G. L. Nemhauser, and G. Sigismondi. The fleet assignment problem: Solving a large-scale integer program. *Mathematical Programming*, 70(1):211–232, 1995.

E. He, N. Boland, G. Nemhauser, and M. Savelsbergh. Dynamic discretization discovery algorithms for time-dependent shortest path problems. *Optimization Online*, 2019.

E. He, N. Boland, G. Nemhauser, and M. Savelsbergh. Time-dependent shortest path problems with penalties and limits on waiting. *INFORMS Journal on Computing*, 2020.

M. Hewitt. Enhanced dynamic discretization discovery for the continuous time load plan design problem. *Transportation Science*, 53(6):1731–1750, 2019.

K. Holmberg and J. Hellstrand. Solving the uncapacitated network design problem by a lagrangean heuristic and branch-and-bound. *Operations research*, 46(2):247–259, 1998.

S. Ichoua, M. Gendreau, and J.-Y. Potvin. Vehicle dispatching with time-dependent travel times. *European journal of operational research*, 144(2):379–396, 2003.

O. Jabali, T. Van Woensel, and A. De Kok. Analysis of travel times and co2 emissions in time-dependent vehicle routing. *Production and Operations Management*, 21(6):1060–1074, 2012.

A. I. Jarrah, J. Goodstein, and R. Narasimhan. An efficient airline re-fleeting model for the incremental modification of planned fleet assignments. *Transportation Science*, 34(4):349–363, 2000.

D. S. Johnson and M. R. Garey. *Computers and intractability: A guide to the theory of NP-completeness*. WH Freeman, 1979.

E. Kanoulas, Y. Du, T. Xia, and D. Zhang. Finding fastest paths on a road network with speed patterns. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 10–10. IEEE, 2006.

D. Kim, C. Barnhart, K. Ware, and G. Reinhardt. Multimodal express package delivery: A service network design application. *Transportation Science*, 33(4):391–407, 1999.

F. Lagos, N. Boland, and M. Savelsbergh. Dynamic discretization discovery for solving the continuous time inventory routing problem with out-and-back routes.

A. Land. Doig. ag,"an automatic method for solving discrete programming problems". *Econometrica*, 28(3):497, 1960.

J. Letchner, J. Krumm, and E. Horvitz. Trip router with individualized preferences (trip): Incorporating personalization into route planning. In *Proceedings of the National Conference on*

Artificial Intelligence, volume 21, page 1795. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

L. Li, W. Hua, X. Du, and X. Zhou. Minimal on-road time route scheduling on time-dependent graphs. *Proceedings of the VLDB Endowment*, 10(11):1274–1285, 2017.

F. Liberatore, G. Righini, and M. Salani. A column generation algorithm for the vehicle routing problem with soft time windows. *4OR*, 9(1):49–82, 2011.

T. L. Magnanti and R. T. Wong. Network design and transportation planning: Models and algorithms. *Transportation science*, 18(1):1–55, 1984.

L. Marshall. Private communication. January 2019.

L. Marshall, N. Boland, M. Savelsbergh, and M. Hewitt. Interval-based dynamic discretization discovery for solving the continuous-time service network design problem. *Transportation Science*, 2020.

L. Marshall, N. Boland, M. Savelsbergh, and M. Hewitt. Interval-based dynamic discretization discovery for solving the continuous-time service network design problem. *Transportation Science*, 55(1):29–51, 2021.

D. McDaniel and M. Devine. A modified benders' partitioning algorithm for mixed integer programming. *Management Science*, 24(3):312–319, 1977.

K. Nachtigall. Time depending shortest-path problems with applications to railway networks. *European Journal of Operational Research*, 83(1):154–166, 1995.

G. L. Nemhauser. Column generation for linear and integer programming. *Optimization Stories*, 20 (65):U73, 2012.

M. Nykänen and E. Ukkonen. The exact path length problem. *Journal of Algorithms*, 42(1):41–53, 2002.

J. Omer and M. Poss. A polynomial algorithm for minimizing travel time in time-dependent networks with waits. 2019a.

J. Omer and M. Poss. Time-dependent shortest paths with discounted waits. *Networks*, 2019b.

A. Orda and R. Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of the ACM (JACM)*, 37(3):607–625, 1990.

M. Padberg and G. Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations research letters*, 6(1):1–7, 1987.

D. F. Rogers, R. D. Plante, R. T. Wong, and J. R. Evans. Aggregation and disaggregation techniques and methodology in optimization. *Operations Research*, 39(4):553–582, 1991.

Y. O. Scherr, M. Hewitt, B. A. N. Saavedra, and D. C. Mattfeld. Dynamic discretization discovery for the service network design problem with mixed autonomous fleets. *Transportation Research Part B: Methodological*, 141:164–195, 2020.

S. Sivanandam and S. Deepa. Genetic algorithm optimization problems. In *Introduction to genetic algorithms*, pages 165–209. Springer, 2008.

J. P. Sousa and L. A. Wolsey. A time indexed formulation of non-preemptive single machine scheduling problems. *Mathematical programming*, 54(1):353–367, 1992.

J. M. van den Akker, C. A. Hurkens, M. W. Savelsbergh, et al. *A time-indexed formulation for single-machine scheduling problems: branch-and-cut*. Université Catholique de Louvain. Center for Operations Research and . . . , 1996.

P. H. Vance, C. Barnhart, E. L. Johnson, and G. L. Nemhauser. Solving binary cutting stock problems by column generation and branch-and-bound. *Computational optimization and applications*, 3(2):111–130, 1994.

D. M. Vu, M. Hewitt, N. Boland, and M. Savelsbergh. Dynamic discretization discovery for solving the time-dependent traveling salesman problem with time windows. *Transportation Science*, 54(3):703–720, 2020.

X. Wang and A. C. Regan. Local truckload pickup and delivery with hard time window constraints. *Transportation Research Part B: Methodological*, 36(2):97–112, 2002.

X. Wang and A. C. Regan. On the convergence of a new time window discretization method for the traveling salesman problem with time window constraints. *Computers & Industrial Engineering*, 56(1):161–164, 2009.

Y. Wang and N. L. Nihan. Freeway traffic speed estimation with single-loop outputs. *Transportation Research Record*, 1727(1):120–126, 2000.

W. E. Wilhelm. A technical review of column generation in integer programming. *Optimization and Engineering*, 2(2):159–200, 2001.

D. P. Williamson and D. B. Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

Windstream Enterprise Retail. The future of unified commerce. https://www.windstreamenterprise.com/wp-content/uploads/2020/08/the-future-of-unified-commerce-insights-paper.pdf, 2020.

H. Wu, I. Herszterg, M. Savelsbergh, and Y. Huang. Service network design for same-day delivery with hub capacity constraints. *Optimization Online*, 7675, 2020.